



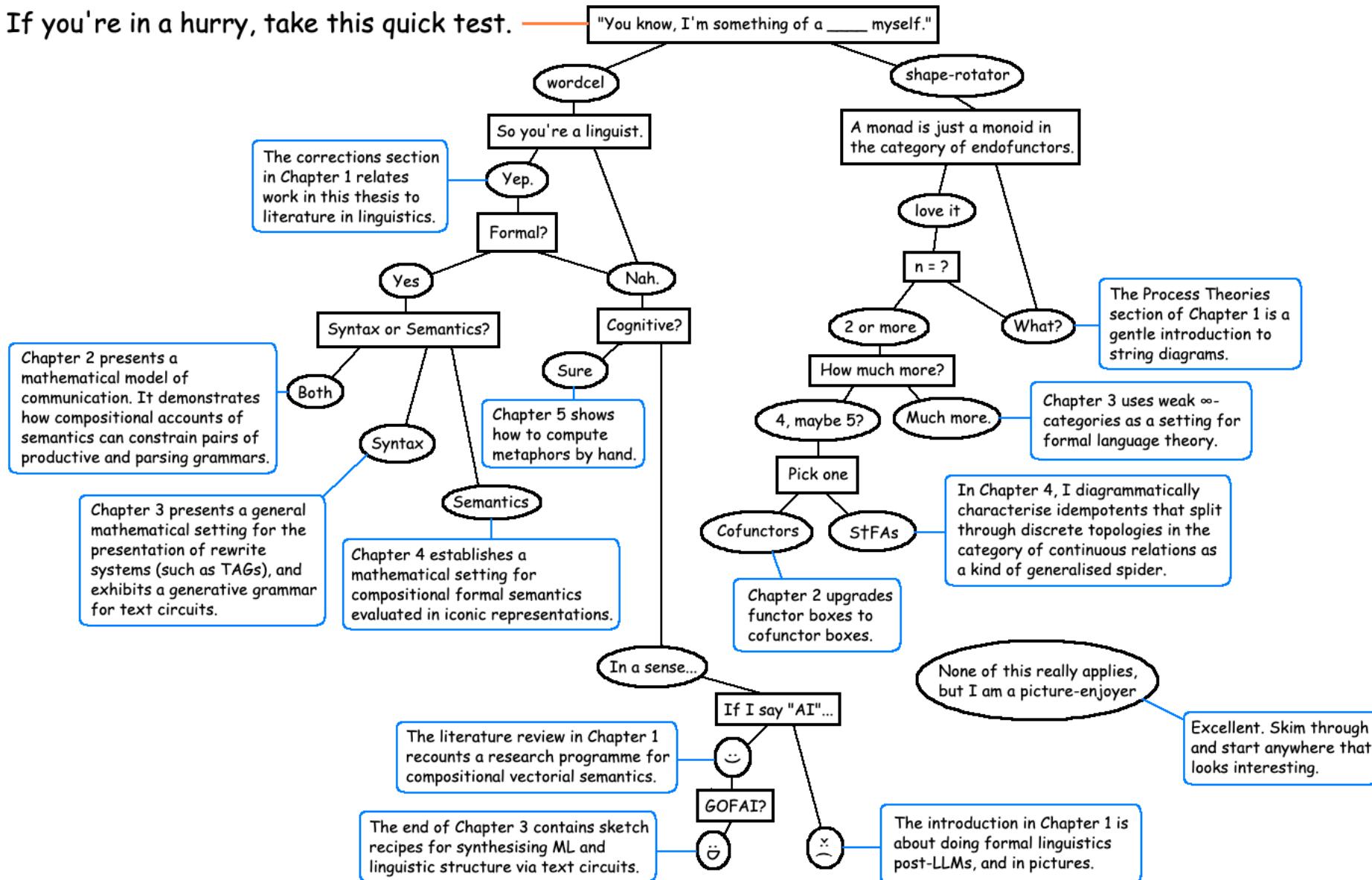
STRING DIAGRAMS FOR TEXT

VINCENT WANG-MASCIANICA

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

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
2023

If you're in a hurry, take this quick test.



Contents

1	Context and synopsis	9
1.1	What this thesis is about	10
1.2	Question: What is the practical value of studying language when Large Language Models exist?	12
1.3	First Reply: Interpretability, maybe.	13
1.3.1	Objection: You're forgetting the bitter lesson.	16
1.3.2	Objection: GOFAI? GO-F-yourself!	16
1.3.3	Objection: How does any of this improve capabilities?	17
1.4	Second Reply: LLMs don't help us understand language; how might string diagrams help?	18
1.4.1	Objection: Isn't the better theory the one with better predictions?	18
1.4.2	Why Category Theory?	20
1.4.3	Objection: Aren't string diagrams just graphs?	21
1.5	Synopsis of the thesis	22
1.6	Process Theories	24
1.6.1	What does it mean to copy and delete?	27
1.6.2	What is an update?	29
1.7	Previously, on DisCoCat	35
1.7.1	Lambek's Linguistics	35
1.7.2	Coecke's Composition	38
1.7.3	Categorical quantum mechanics	40
1.7.4	Enter computational linguistics	42
1.7.5	I killed DisCoCat, and I would do it again.	46
1$\frac{1}{2}$	Corrections	57
1 $\frac{1}{2}$.1	Does it work?	57
1 $\frac{1}{2}$.1.1	Have experiments been done?	58

$1\frac{1}{2}.2$	Can you situate this work with respect to the literature in formal linguistics?	58
$1\frac{1}{2}.2.1$	On Pregroups to Text Circuits vs. Transformational Grammar to Formal Semantics	59
$1\frac{1}{2}.2.2$	On Montague's conception of grammar	60
$1\frac{1}{2}.2.3$	On Deep Structure, the Universal Base Hypothesis, and the "Lexical Objection"	63
$1\frac{1}{2}.2.4$	On communication, and the mathematical infeasibility of the Autonomy of Syntax	69
$1\frac{1}{2}.2.5$	On frameworks for rewriting systems	72
$1\frac{1}{2}.2.6$	On formality in cognitive semantics	73
2	On communication	75
2.1	How do we communicate using language?	76
2.1.1	An issue with functorial semantics of internal wirings	84
2.2	Discrete Monoidal Opfibrations	86
2.2.1	What are they good for?	92
2.3	Strictified diagrams for monoidal categories	94
2.4	Monoidal cofunctor boxes	99
2.5	Monoidal kinda-cofunctor boxes	102
2.6	Discussion and Limitations	109
3	Text circuits for syntax	115
3.1	An introduction to weak n-categories for formal linguists	116
3.1.1	String-rewrite systems as 1-object-2-categories	117
3.1.2	Tree Adjoining Grammars	119
3.1.3	Tree adjoining grammars with local constraints	126
3.1.4	Braiding, symmetries, and suspension	126
3.1.5	TAGs with links	131
3.1.6	Full TAGs in weak n -categories	134
3.2	A generative grammar for text circuits	136
3.2.1	A circuit-growing grammar	136
3.2.2	Text circuit theorem	149
3.3	Text circuits: details and development	160
3.4	How to play	161
4	Continuous relations for semantics	177
4.1	Continuous Relations for iconic semantics	178
4.2	Continuous Relations by examples	180

4.3	The category ContRel	186
4.3.1	Symmetric Monoidal structure	186
4.3.2	Rig category structure	187
4.3.3	Monoidal (co!)closure	188
4.3.4	Category-theoretic endnotes	192
4.4	Populating space with shapes using sticky spiders	197
5	Sketches in iconic semantics	223
5.1	Preliminary concepts for the sketches	224
5.1.1	Open sets: concepts	224
5.1.2	Using sticky spiders as location-tests	225
5.1.3	Copy: stative verbs and adjectives	226
5.1.4	The unit interval	227
5.1.5	Metric structure	231
5.1.6	Relational homotopy	233
5.1.7	Coclosure: adverbs and adpositions	235
5.1.8	Nice spiders	236
5.2	Composition of dynamic verbs via temporal anaphora	237
5.3	Iconic semantics for modal verbs	242
5.4	Iconic semantics for general anaphora via Turing objects	246
5.5	Configuration spaces	254
5.6	Formal models of figurative language	259
5.6.1	Temperature and colour: the Planckian Locus	260
5.6.2	Time and Money: complex conceptual structure	263
5.7	A specimen problem sheet from an imaginary future	268
6	Bibliography	275

Acknowledgements To my parents and siblings: thank you for supporting me. I don't know how I can ever make it all up to you, but I'll try my best. I love you.

To my academic parents, Bob and Philipp. Thank you for your tutelage. I am glad to have found brothers in life such as you, and I hope we can tell many more stories together.

To my seniors — Dan and Konstantinos here at Oxford; Jamie in Cambridge, Dusko through our letters; Salvadore, Beau and Sean throughout our time at LP-PRD; Luciano and Rosaria at those wonderful meetings at the OII; Martha and Mehrnoosh and Moortgat at ESSLI; David and David and Brendan at Topos in sunny California; Pawel and Amar in (less sunny) Estonia; and Tim from across the desk-castle. You walked far along the paths I am still only just setting out on. Thank you for sharing your lessons with me.

A special thanks for you, Jules and Stefano. You actually read the whole damn thing and sat with me through six and a half hours of viva. It was an honour to get doctored by you.

To my cohort, both in Oxford — Nick, Nicola, Lukas, Guillaume, Cole, Sivert, Nihil, James, Matt — and elsewhere — Mario, Elena, Chad, Nathan, Bruno, Matteo, Eigel — COVID made for an unusual DPhil, but I wouldn't have had it any other way because of you. Thank you for teaching me so much.

To Jono, Ben, Razin, Lia, Hamza, and Tiffany. I'm sorry for whatever part I played in getting you into this language business, it seemed like a good idea at the time. I hope at least it's a springboard towards better things. For Jono in particular: thanks for qualming.

To Aleks, Ilyas, and the little demon who helps me convert various drugs into diagrams and text at the cost of my health: thank you for the environments you have looked after, within which this thesis was made logically possible.

To my future council of godfathers for my son Felix: Andrew, Colin, Maxim, Will, and Joel. Thank you for putting up with me and keeping me grounded. Paulina, my love. Thank you for sticking with me and helping me grow. Nothing is as important to me as being happy together with you.

Novel contributions:

- Section 3.1 is a pedestrian introduction to weak n -category theory (via `homotopy.io`, underpinned by the theory of associative n -categories) from the perspective of generalising familiar string-rewrite systems to higher dimensions. The chief development of this section is a demonstration that context-free grammars and tree-adjoining grammars may be formalised in the n -categorical setting.
- Section 3.2 spells out a generative grammar for text using an n -categorical signature as a rewrite system, which additionally provides a unified framework from which the Text Circuit Theorem first proved in [WLC23] is recovered.
- Section 4.1 introduces the category **ContRel** of continuous relations. I detail the relationships (or lack thereof) of **ContRel** to its cousins **Top** and **Rel** in Section 4.3. Though **ContRel** is constructed naïvely, its definition and an exposition of its expressivity from the monoidal perspective appears to be novel.
- Section 4.4 string-diagrammatically characterises set-indexed collections of disjoint open subsets of spaces in **ContRel** as *sticky spiders* – special frobenius algebras that satisfy certain interaction relations with an idempotent. The diagrammatic outcome is that reasoning with such set-indexed collections remains as graphically intuitive as with spiders.
- Section 2.1 argues for the centrality of explaining communication as a criterion for formal approaches to syntax, and explores the relationship between productive and parsing grammars as organised by a monoidal cofunctor. A diagrammatic treatment of monoidal cofunctor boxes is introduced for this purpose.
- In Section 5.6, by parsing text as circuits (Section 3.2) and using merge-boxes (Section 2.1) to interpret those circuits in **ContRel** as iconic representations, I show how one may compute metaphors by hand.

And there are a couple of odds and ends in the corrections and the sketches. The diagrams in this work were mostly created using TikZiT, and they are available at <https://github.com/vinnylarouge/Thesis>. If you have comments, suggestions, or corrections, my email is vincentwangsemailaddress@gmail.com

1

Context and synopsis

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

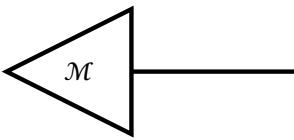


Figure 1.1: Let's say that the meaning of text is how it updates a model. So we start with some model of the way things are, modelled as data on a wire.

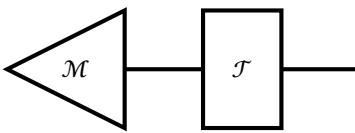


Figure 1.2: Text updates that model; like a gate updates the data on a wire.

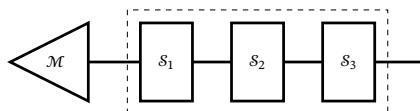


Figure 1.3: Text is made of sentences; like a circuit is made of gates and wires.

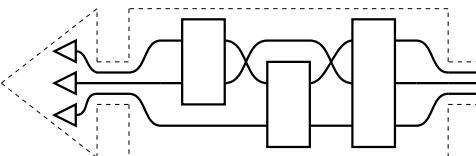


Figure 1.4: Let's say that *The meaning of a sentence is how it updates the meanings of its parts*. As a first approximation, let's say that the *parts* of a sentence are the nouns it contains or refers to. Noun data is carried by wires. Collections of nouns are related by gates, which play the roles of verbs and adjectives.

1.1 What this thesis is about

THIS THESIS IS ABOUT STUDYING LANGUAGE USING STRING DIAGRAMS.

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

OBJECTION: ISN'T THAT REINVENTING THE WHEEL?

Yes, to an extent. I am not interested in the human language faculty *per se*, so my aims differ. There are several potential practical and theoretical benefits that a fresh mathematical perspective on language enables. First, the mathematics of applied category theory allows us to unify different views of syntax, and conservatively generalise formal semantics to aspects of language that may have seemed beyond the reach of rigour, such as metaphor. Practically, the same mathematics allows us to construct interfaces between syntax/structure and semantics/implementation in such a way that we can control the former and delegate the latter by providing specifications without explicit implementation, which (for historical reasons I will explain shortly) is possibly the least-bad idea for getting at natural language understanding in computers from the bottom-up. Second, there are probably benefits to expressing linguistics in the same mathematical and diagrammatic lingua franca that can be used to represent and reason – often soundly and completely – about linear and affine algebra [Sob15, BSZ17, BPSZ19], first order logic [HS20, BDGHS24], causal networks [LT23, JKZ19], signal flow graphs [BSZ14], electrical circuits [BS22], game theory [Hed15], petri nets [BM20], probability theory [FGP21], machine learning [CGG⁺22, KLLW24, RFL⁺24], and quantum theory [CK17, CG23], to name a few applications. At the moment, the practical achievements of language algorithms de-emphasise the structure of language, and there is no chance of reintroducing the study of structure with dated mathematics.

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

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

POINT OF INFORMATION: WHAT ARE STRING DIAGRAMS?

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

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

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

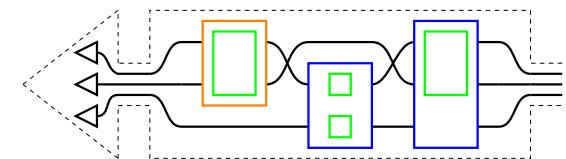


Figure 1.5: Gates can be related by higher order gates, which play the roles of adverbs, adpositions, and conjunctions; anything that modifies the data of first order gates like verbs.

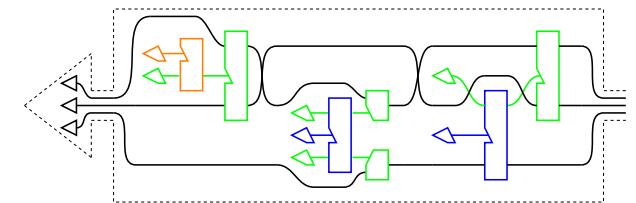


Figure 1.6: In practice, higher order gates may be implemented as gates that modify parameters of other gates. Grammar, and *function words* – words that operate on meanings – are in principle absorbed by the geometry of the diagram. These diagrams are natural vehicles for *dynamic semantics* [NBvV22], broadly construed, where states are prior contexts and sentences-as-processes update prior contexts.

Definition 1.1.1 (Text Circuits). *Text circuits* are made up of three ingredients:

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

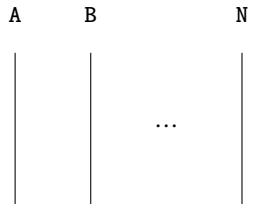


Figure 1.7: Nouns are represented by wires, each ‘distinct’ noun having its own wire.

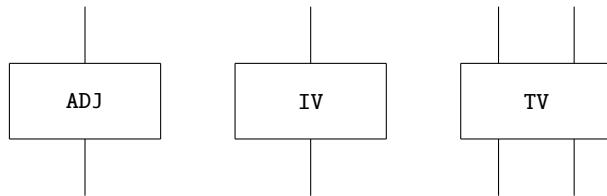


Figure 1.8: We represent adjectives, intransitive verbs, and transitive verbs by gates acting on noun-wires. Since a transitive verb has both a subject and an object noun, that will then be two noun-wires, while adjectives and intransitive verbs only have one.

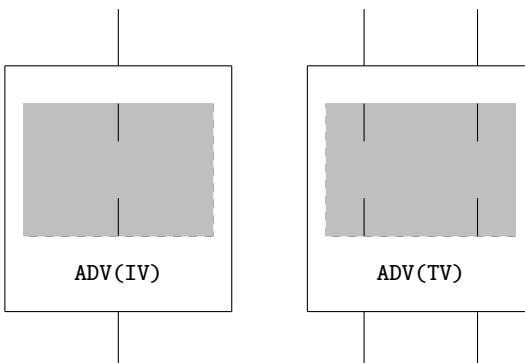


Figure 1.9: Adverbs, which modify verbs, we represent as boxes with holes in them, with a number of dangling wires in the hole indicating the shape of gate expected, and these should match the input- and output-wires of the box with the whole.

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

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

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

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

1. LLMs are a civilisational milestone technology. A force-multiplication tool for natural language – the universal interface – built from abundant data and compute in the information age may have comparably broad, deep, and lasting impact to the conversion of abundant chemical fuel to physical energy by steam engines in the industrial revolution.
2. LLMs represent a paradigm shift for humanity because they threaten our collective self-esteem, in a more pointed manner than losing at chess or Go to a computer; modifying a line of thinking from [Flo14], LLMs demonstrate that language and (the appearance of) complex thought that language facilitates is not a species-property for humans, and this stings on par with Darwin telling us we are ordinary animals like the rest, or Galileo telling us our place in the universe is unremarkable.
3. LLMs embody the latest and greatest case study of the bitter lesson [Sut19]. The tragedy goes like this: there's a group of people who investigate language – from syntax and semantics to pragmatics and analogies and storytelling and slang – who treat their subject with formal rigour and have been at it for many centuries. Their role in the story of LLMs is remarkable because it doesn't exist. They were the only qualified contestants in a "let's build a general-purpose language machine" competition, and they were a no-show. Now the farce: despite the fact that all of their accumulated understanding and theories of language were left out of the process, the machine is not only built but also far exceeds anything we know how to build in a principled way out of all their hard-earned insight. That is the bitter lesson: dumb methods that use a lot of data and compute outperform clever design and principled understanding.

1.3 *First Reply: Interpretability, maybe.*

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

POINT OF INFORMATION: WHAT IS THE POVERTY OF THE STIMULUS? In short, this famous problem is the observation that humans learn language despite having very little training data, in comparison to the complexity of the learned structure. It is on the basis of this observation – alongside many others surrounding language acquisition and use – that Chomsky posits [Cho00] that language is an innate human faculty, the

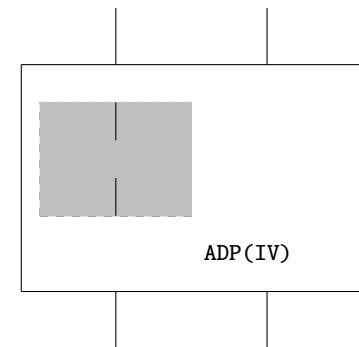


Figure 1.10: Similarly, adpositions also modify verbs, by moreover adding another noun-wire to the right.

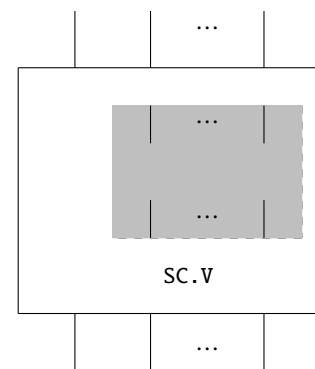


Figure 1.11: For verbs that take sentential complements and conjunctions, we have families of boxes to accommodate input circuits of all sizes. They add another noun-wire to the left of a circuit.

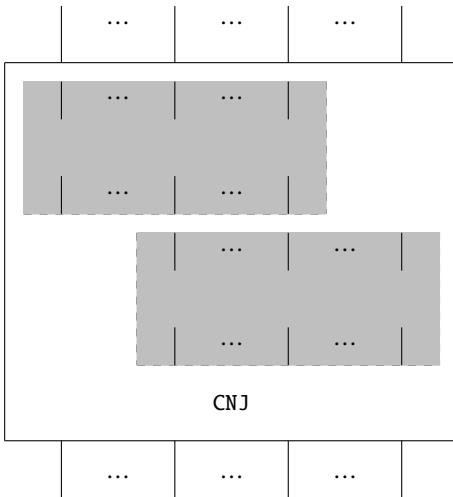


Figure 1.12: Conjunctions are boxes that take two circuits which might share labels on some wires.

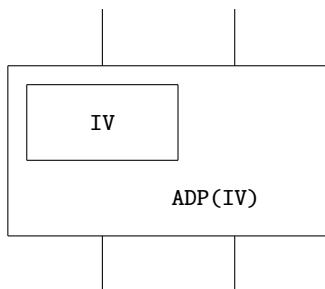


Figure 1.13: Of course filled up boxes are just gates.

development of which is less like effortfully going to the gym and more like effortlessly growing arms you were meant to have. The explanation goes like this: we can explain how a complex structure like grammar gets learnt from a small amount of data if everyone shares an innate Universal Grammar with a small number of free parameters to be learned. Whether or not the intermediate mechanism is a species-property of humans, the point is that we humans get a very small amount of input data, that data interacts with the mechanism in some way, and then we know a language. So, now that there are language-entities that are human-comparable in competence, we can make a back-of-the-envelope approximation of how much work the intermediate mechanism is doing or saving by comparing the difference in how much data and compute is required for both the human and for the machine to achieve language-competence. Humans get about 1.5 megabytes of data [MP19], 90 billion neurons [Her12], and an adult human consumes around 500 calories per day for thinking, for let's say 20 years of language learning. Rounding all values *up* to the closest order of magnitude, this comes to a cost metric of 10^{29} bits \times joules \times neurons. PaLM – an old model which is by its creators' account the first language model to be able to reason and joke purely on the basis of linguistic ability and without special training [CND⁺22, NC22] – required 780 billion training tokens of natural language (let's discount the 198 gigabytes of source code training data), which we generously evaluate at a rate of 4 characters per token [Kha23] and 5 bits per character. The architecture has 540 billion neurons, and required 3.2 million kilowatt hours of energy for training [Tom22]. Rounding values for the three units down *down* to the nearest order of magnitude comes to a cost metric of 10^{41} bit-joule-neurons. Whatever the human mechanism is, it is responsible for an order of magnitude in efficiency *give or take an order of magnitude of orders of magnitude*. It's possible that over time we can explain this difference away by various factors such as the efficiency of meat over minerals, separating knowledge of the world from knowledge of language, more efficient model architectures, or the development of efficient techniques to train new language models using old ones [TGZ⁺23]. One thing is clear: if it is worth hunting a fraction of a percent of improvement on a benchmark, forget your hares, a 10^{10} factor is a stag worth cooperating for.

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

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

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

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

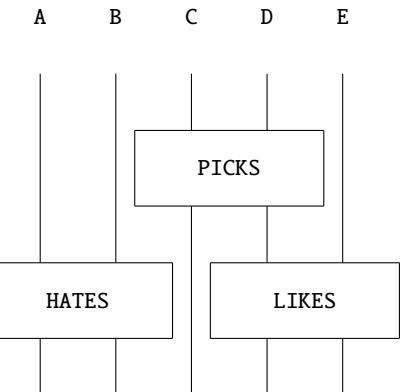


Figure 1.14: Gates compose sequentially by matching labels on some of their noun-wires and in parallel when they share no noun-wires, to give text circuits.

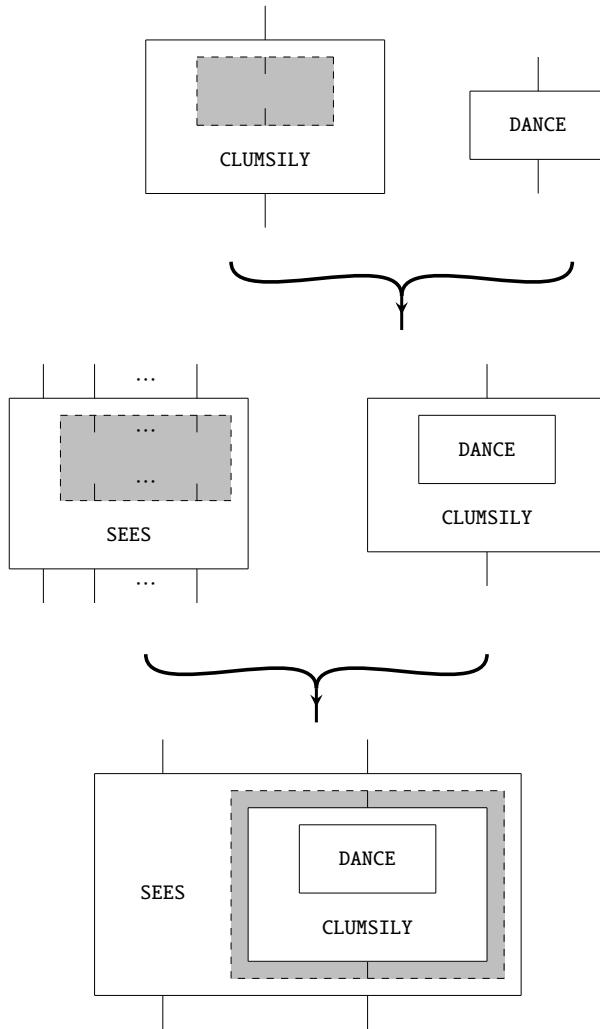


Figure 1.15: To summarise: composition by nesting corresponds to grammatical structure within sentences. Sentences correspond to filled gates, boxes with fixed arity correspond to first-order modifiers such as adverbs and adpositions, and boxes with variable arity correspond to sentential-level modifiers such as conjunctions and verbs with sentential complements.

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

The bitter lesson is so harsh and often-enough repeated that this viewpoint is worth addressing proactively. The caveat that saves us is that the curse of expertise applies only to the object-language of the problem to be solved, not model architectures. We agree that qualitative improvements in problem-solving ability rarely if ever arise from encoding expert knowledge of the problem domain. Instead, these improvements come from *architectural* innovations, which means altering the parts and internal interactions of a model: changing *how* it thinks rather than *what* it thinks, to paraphrase Sutton's original prescription. We have good historical evidence that this prescription works, which we see by tracing the evolutionary path for data-driven language models from markov chains to deep learning [LBH15], RNNs [RM87], LSTMs [HS97], and now transformers [VSP⁺17]. Such structural changes are motivated by understandings (at varying degrees of formality) of the "geometry of the problem" [BBCV21]. The value proposition here is that with an appropriate mathematical lingua franca for structure, composition, and interaction, we can mindfully design rather than stumble upon the "meta-methods" Sutton calls for, allowing experts to encode *how* machines think and discover rather than *what*. Importing compositional and structural understanding from linguistics to machine learning via string diagrams might allow us to cheat the bitter lesson in spirit while adhering to the letter, and there is some preliminary empirical evidence for this, which I report on in Section 3.3.

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

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

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

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

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

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

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

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

To distinguish the difference in approach here, I have to draw a distinction in neurosymbolic approaches that does not seem well-supported in the literature. There are many approaches concerned with using connectionist architectures to simulate or aid or be-aided-by symbolic composition, which we can see the beginnings of in LLMs by examples such as chain-of-thought reasoning [WWS⁺23], and by probing their behaviour with respect to understood symbolic models [KW23]. The second kind of approach I would like to articulate is the inverse, where connectionist architectures are organised and reasoned with by symbolic-compositional means. Some examples of the first kind include implementing data structures as operations on high-dimensional vectors, taking advantage of the idiosyncrasies of linear algebra in very high dimension [Kan19], or work that explores how the structure of word-embeddings in latent space encode semantic relationships between tokens. Some examples of the second kind include reasoning about the capability of graph

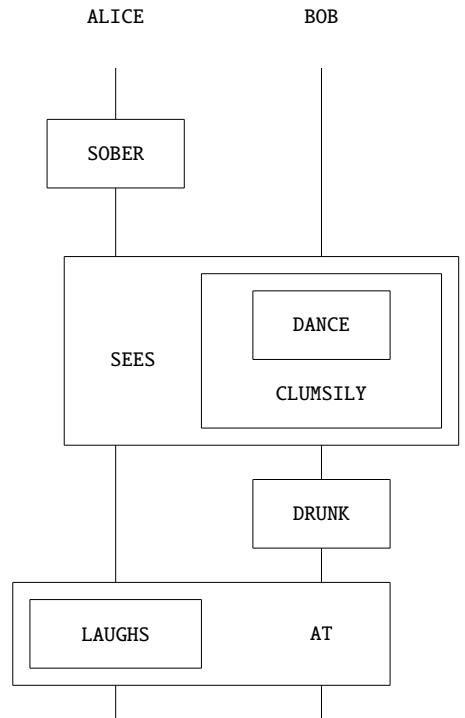


Figure 1.16: Composition by connecting wires corresponds to identifying coreferences in discourse. We obtain the same circuit for multiple text presentations of the same content, e.g. Sober Alice who sees drunk Bob clumsily dance laughs at him. yields the same circuit as the text Alice is sober. She sees Bob clumsily dance. Bob is drunk. She laughs at him.

I recount the following from [Søg23], which argues that symbol-grounding is solvable from data alone, and in the process surveys the front of the symbol-grounding problem in AI: the issue of whether LLMs encode what words refer to and mean. On the account of [BK20], the performance of current LLMs is a form of Chinese Room [Sea80] phenomenon, so no amount of linguistic competence can be evidence that LLMs solve the symbol-grounding problem. However, the available evidence appears to suggest otherwise. For example, large models converge on word embeddings for geographical place names that are isomorphic to their physical locations [LAS21]. Since we know that brain activity patterns encode abstract conceptual space with the same mechanisms as they do physical spaces [KS16], extrapolating the ability of LLMs to encode spatially-analogical representations would in the limit suggest that LLMs encode meanings in a way isomorphic to how we do, modulo the token-word distinction and so long as we take seriously some version of Gärdenfors' [Gär14] thesis that meaning is encoded geometrically.

neural networks by identifying or isolating their underlying compositional structure [LGT23], or architectures whose behaviour arises from compositional structure using neural nets as constituent parts, such as GANs [GPM⁺14] and gradient boosted decision trees [CG16]. The work in this thesis builds upon a research programme – DisCoCat [CSC10], elaborated in Section 1.7 – which lies somewhere in the middle of a duality of approaches to merging connectionism and symbolic-composition. It is, to the best of my knowledge, the only approach that explicitly incorporates mathematically rigorous compositional structures from the top-down alongside data-driven learning methods from the bottom-up. Fortifying this bridge across the aisle requires a little give from both sides; I ask only that reader entertain some pretty string diagrams.

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

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

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

LLMs are a theory of language in the same way a particular human brain is a theory of cognition; at best, debatably. There are various criteria – not all independent – that are arguably necessary for something to qualify as an explanatory theory, and while LLMs suffice (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 contrast to the idealised and partial nature of formal theories, the nature of LLMs is that they are trained on empirical data about language

that captures the friction of the real world. So, in terms of raw predictive power, we should naturally expect the LLMs to have an advantage over principled theories. They are so good at empirical capture that to some degree they automatically satisfy the related criteria of coherence – consistency with other established linguistic theories – and scope – the ability to capture a wide range of phenomena. But while empirical capture is necessary for explanatory theories, it is insufficient.

There are several criteria where the adequacy of LLMs is unclear or debatable. Fruitfulness is a sociological criterion for goodness of explanatory theories, in that they should generate new predictions and lead to further discoveries and research. While they are certainly a potent catalyst for research in many fields even beyond machine learning, it is unclear for now how they relate to the subject matter of linguistics. Whether they satisfy Popper's criterion of falsifiability is as of yet not determined, because it is not settled how to go about falsifying the linguistic predictions of LLMs, or even express what the content of a theory embodied by an LLM is. The closest examples to falsifiability that come to mind are tests of LLM fallibility for reasoning and compositional phenomena [DLS⁺23], or their weakness to adversarial prompt-injections [Ril22], but these weaknesses do not shed light on their linguistic competence and "understanding" directly.

Now the disappointments. As far as we can tell; LLMs are far from simple, and simplicity (Occam's Razor) is an ancient criterion for the goodness of explanation; while they exhibit, they do not explain the structure, use, and acquisition of language; they do not unify or subsume our prior understanding of linguistics. The first two points are basically unobjectionable, so I will briefly elaborate on the criterion of unification and subsumption of prior understandings, borrowing a framework from cognitive neuroscience. A common methodology for investigating cognitive systems is Marr's 3 levels [Mar10] (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 one 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 calculator or as values, reads, and writes in computer memory. In the case of LLMs, we understand well the nature of the computational theory level, at least in their current incarnation as next-token-predictors, which is a narrow and clear task. Furthermore, we understand the hardware level well, from the silicon going up through the ladder of abstraction to software libraries and the componentwise activity of neural nets. Yet somehow, we know everything and nothing at once about the representation and algorithm level; we can explain how transformer models work in terms of attention mechanisms and lookback, and how it is that these models are trained using data to produce the outputs they do. In spite of understandings which should jointly cover all of level 2, we cannot relate their operations on language to our own.

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

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

1.4.2 Why Category Theory?

THE SHORT ANSWER: NO REASON. Implicitly, what's wrong with λ -calculus and whatever else? The short answer is that there's no reason to use category theory if you don't feel like it's worth the effort. It's definitely not an issue of expressivity: after all, whatever we can do with a modern programming language we can also do with punchcards in principle, and one can think of category theory as just a high-level math language that abstracts away a lot of details some may consider unimportant.

THE LONGER ANSWER: WHY NOT? The modeller mediates the gap between mathematics and reality by a necessarily subjective process. If formal linguistics is a hobby, then the choice of mathematics used is merely a matter of taste, and there is no need for further discussion. If however formal – explicitly *mathematical* – linguistics aspires to something universal and canonical, then it may be a good idea to start with a mathematical metalanguage where structural similarities, compositionality, and modularity are primitives. Now let me sketch why using some more complicated mathematics might in this case be a good idea.

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

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

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

different ways to encode an ordered pair. Kuratowski's definition is

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

Which could have just as easily been:

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

And here is Wiener's definition:

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

But we don't care what the precise implementation is so long as the property that $(a, b) = (c, d)$ just when $a = c$ and $b = d$ holds. The same kind of problem keeps occurring at all levels of complexity: suppose you have a set-indexed set of things $\{T_i \mid i \in I\}$, which you can choose to implement as a function $I \rightarrow \mathbf{T}$. Then somebody else wants to make the indexing set dynamically updatable with novel elements, so they have to rephrase the indexing mechanism as a set of tuples $\{(T_{i^1}, i^1), (T_{i^2}, i^2), \dots\}$ so that they can add or remove elements, and then someone else comes along and decides that the indexes have structure that disallow certain things to be indexed... All this means that if one wants to use set theory to relate different theories at a "structural" level, one must first analyse both in terms of their constituent sets and functions in order to construct more functions between sets and functions. As you may already know if you're in the business of articulating formal systems, representation-dependency makes this process a bureaucratic nightmare.

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

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

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

The lesson here is that choice of representations matter. In the mathematical context, representations matter because they generalise differently. On the surface, here is an example of two representations of the same platonic mathematical object. However, Tic-Tac-Toe is in the same family as Connect-4 or 5-in-a-row on an

If you are a formal linguist doing serious work with set-theoretic foundations, take this quick test to see if categories and diagrams might be right for you. For each of the statements below, note whether you agree or disagree.

- It is not fun to read, write, or think with set-builder notation.
- It is difficult to relate my work to what other people are interested in.
- It is costly to tinker with and modify my framework.
- It is hard to communicate my framework to others.
- It would be nice to integrate my work with methods used in other fields, like computer science.

If you agreed with any of the above, consult your nearest category theorist to see if string diagrams are right for you. If not, have a nice day.

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

unbounded grid, while the game with numbered cards generalises to different variants of Nim. That they coincide in one instance is a fork in the path. In the same way, viewing string diagrams as "just graphs" is taking the wrong path, just as it would be true but unhelpful to consider graphs "just sets of vertices and edges". String diagrams are indeed "just" a special family of graphs, just as much as prime numbers are special integers and analytic functions are special functions.

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

1.5 Synopsis of the thesis

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

I will develop some diagrammatic technology in Chapter 2, where I introduce monoidal cofunctors for dealing with the kind of systematic relationships we see in language. In the process, I introduce and explain internal wirings, and I also explore how productive and parsing grammars ought to relate to each other in light of the fact that communication is possible.

It will then be necessary to justify some kind of systematic relationship between text circuits and something resembling text in natural language, which will be the purpose of Chapter 3. Here I introduce weak n -categorical signatures as generalisations of string rewrite systems to higher dimensions. I demonstrate that context-free, context-sensitive, and tree-adjoining grammars are all formalisable in this one setting, in which I then construct a generative grammar that simultaneously produces grammatical text as strings of words, and the requisite structure to obtain text circuits. This relationship between text circuits and text will be encapsulated in the Text Circuit Theorem. I close this section with some discussion of ongoing practical and

theoretical developments in text circuits, and point out some avenues of generalisation.

Once we have text circuits, we will need some monoidal category in which to interpret and calculate with them. In particular, it would be nice to calculate formally with the kinds of iconic cartoon representations that are typically used typically as informal schematic illustrations of metaphors. For this purpose, in Chapter 4 I introduce **ContRel**, a symmetric monoidal category of continuous relations. I diagrammatically characterise set-indexed collections of disjoint open subsets of a space – i.e. shapes with labels – as idempotents that interact with a special frobenius algebra. I will then develop a vocabulary of linguistic topological concepts so that shapes can be connected or touching or inside one another, and I will make them move and dance as I please. All of that gets done using just equations between string diagrams, and the diagrams underpinning these iconic semantics are a natural basis upon which one can perform truth-conditional analyses.

Then we will have text circuits, a formal setting to reason with and about cartoons, and diagrammatic techniques to form a structured correspondence between a text and its representation as a cartoon, at which point I will do a couple of sketches in Chapter 5, and close with the computation of a metaphor.

1.6 Process Theories

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

Definition 1.6.1 (Category). A *category* \mathcal{C} consists of the following data

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

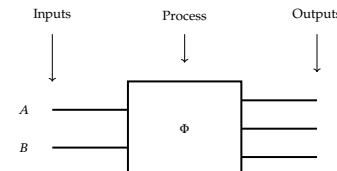
This data has to satisfy two coherence conditions, which are:

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

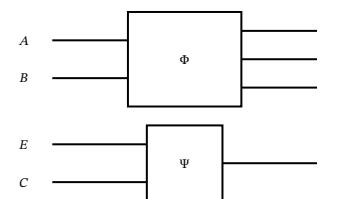
Associativity: For any four objects A, B, C, D and three morphisms $f : A \rightarrow B, g : B \rightarrow C, h : C \rightarrow D$, $(f; g); h = f; (g; h)$

This section seeks to introduce process theories via string diagrams. The margin material will provide the formal mathematics of string diagrams from the bottom-up. The main body develops process theories via string diagrams by example, through which we develop towards a model of linguistic spatial relations – words like "to the left of" and "between" – which are a common ground of competence we all possess. Here we only focus on geometric relations between points in two dimensional Euclidean space equipped with the usual notions of metric and distance, providing adequate foundations to follow [WC21], in which I demonstrate how text circuits can be obtained from sentences and how such text circuits interpreted in the category of sets and relations **Rel** provides a semantics for such sentences. This motivates the question of how to express the (arguably more primitive [Jea67]) linguistic topological concepts – such as "touching" and "inside", which we provide all the necessary tools for in Section 5.1. We close this section with a brief note on how process theories relate to mathematical foundations and computer science.

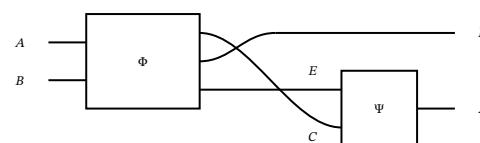
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. Unless otherwise specified, we read processes from left to right.



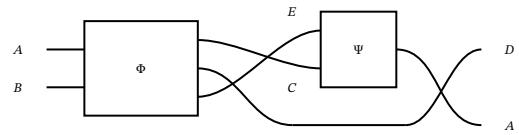
Processes may compose in parallel, depicted as placing boxes next to each other.



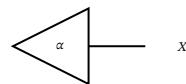
Processes may compose sequentially, depicted as connecting wires of the same type.



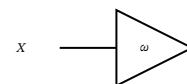
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



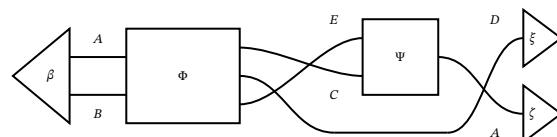
Some processes have no inputs; we call these *states*.



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



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



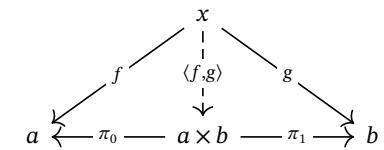
A process theory is given by the following data:

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

Example 1.6.8 (Linear maps with direct sum). Systems are finite-dimensional vector spaces over \mathbb{R} . Processes are linear maps, expressed as matrices with entries in \mathbb{R} .

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

Definition 1.6.2 (Categorical Product). In a category \mathcal{C} , given two objects $a, b \in \text{Ob}(\mathcal{C})$, the *product* $A \times B$, if it exists, is an object with projection morphisms $\pi_0 : A \times B \rightarrow A$ and $\pi_1 : A \times B \rightarrow B$ such that for any object $x \in \text{Ob}(\mathcal{C})$ and any pair of morphisms $f : X \rightarrow A$ and $g : X \rightarrow B$, there exists a unique morphism $f \times g : X \rightarrow A \times B$ such that $f = (f \times g); \pi_0$ and $g = (f \times g); \pi_1$. This is a mouthful which is easier expressed as a commuting diagram as below. The dashed arrow indicates uniqueness. $A \times B$ is a product when every path through the diagram following the arrows between two objects is an equality.



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

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

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

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

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

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

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

$$\pi_0^W := \{\{a, \emptyset\}, b\} \mapsto a$$

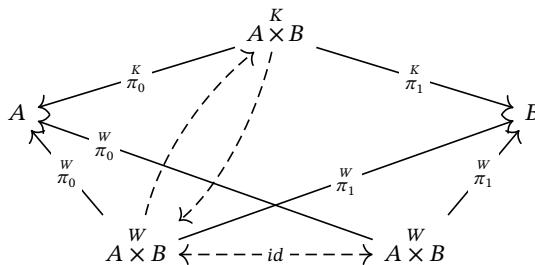
$$\pi_1^W := \{\{a, \emptyset\}, b\} \mapsto b$$

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

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

$$f^W \times g := x \mapsto \{\{f(x)\}, \emptyset, g(x)\}$$

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



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

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

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

Example 1.6.9 (Sets and functions with cartesian product). Systems are sets A, B . Processes are functions between sets $f : A \rightarrow B$. 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 $f \otimes g : A \times C \rightarrow B \times D$ of functions $f : A \rightarrow B$ and $g : C \rightarrow D$ is defined:

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

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

Example 1.6.10 (Sets and relations with cartesian product). Systems are sets A, B . Processes are relations between sets $\Phi \subseteq A \times B$, which we may write in either direction $\Phi^* : A \nrightarrow B$ or $\Phi_* : B \nrightarrow A$. Relations between sets are equivalently matrices with entries from the boolean semiring. Relation composition is matrix multiplication with the boolean semiring. Φ^*, Φ_* are the transposes of one another. Sequential composition is relation composition:

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

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

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

The unit of parallel composition is the singleton. States and tests of a set A are subsets of A .

1.6.1 What does it mean to copy and delete?

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



Example 1.6.11 (Linear maps). Consider a vector space \mathbf{V} , which we assume includes a choice of basis. The copy map for a vector space \mathbf{V} is the rectangular matrix made of two identity matrices:

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

The delete map for \mathbf{V} is an empty column; a matrix of dimensions $\dim(V) \times 0$:

$$\epsilon_{\mathbf{V}} : \mathbf{V} \rightarrow \mathbf{0}$$

Example 1.6.12 (Sets and functions). Consider a set A . The copy function is defined:

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

The delete function is defined:

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

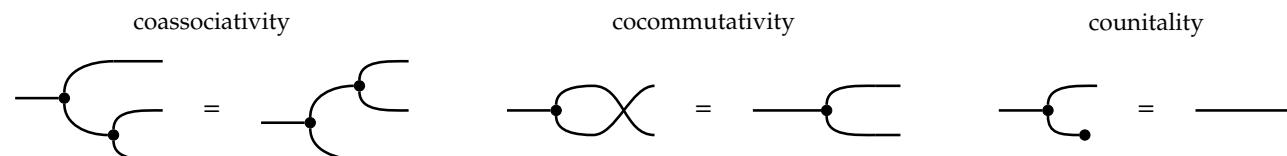
Example 1.6.13 (Sets and relations). Consider a set A . The copy relation is defined:

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

The delete relation is defined:

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

We may verify that, no matter the concrete interpretation of the diagram in terms of linear maps, functions or relations, the following equations characterise a cocommutative comonoid internal to a monoidal category.



It is worth pausing here to think about how one might characterise the process of copying in words; it is

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

Definition 1.6.3 (Product of categories). The product of categories $\mathcal{C} \times \mathcal{D}$ has ordered pairs of objects (C, D) and pairs of morphisms (f, g) with elementwise composition and pairs of identities for identity morphisms. It is also, up to unique isomorphism, the categorical product in \mathbf{Cat} , the category of categories and functors.

Definition 1.6.4 (Functor). A functor $F : \mathcal{C} \rightarrow \mathcal{D}$ (read: with domain a category \mathcal{C} and codomain a category \mathcal{D}) consists of two suitably related functions. An object function $F_0 : \text{Ob}(\mathcal{C}) \rightarrow \text{Ob}(\mathcal{D})$ and a morphism function (equivalently viewed as a family of functions indexed by pairs of objects of \mathcal{C}) $F_1(X, Y) : \mathcal{C}(X, Y) \rightarrow \mathcal{D}(F_0 X, F_0 Y)$. F_1 must map identities to identities – i.e., be such that for all $X \in \mathcal{C}$, $F_1(1_X) = 1_{F_0 X}$ – and F_1 must map composites to composites – i.e., for all $X, Y, Z \in \text{Ob}(\mathcal{C})$ and all $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, $F_1(f; g) = F_1 f; F_1 g$.

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

Definition 1.6.5 (Natural Transformation). A natural transformation $\theta : F \Rightarrow G$ for (co)domain-aligned functors $F, G : \mathcal{C} \rightarrow \mathcal{D}$ is a family of morphisms in \mathcal{D} indexed by objects $X \in \mathcal{C}$ such that for all $f : X \rightarrow Y$ in \mathcal{C} , the following commuting diagram holds in \mathcal{D} :

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

challenging to do so for such an intuitive process. The diagrammatic equations, when translated into prose, provide an answer.

Coassociativity: says there is no difference between copying copies.

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

Countability: 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 $\Delta_X : X \rightarrow X \otimes X$ and $\epsilon_X : X \rightarrow I$ that obey all the equational constraints above, Δ_X and ϵ_X are as good as a copy and delete.

Example 1.6.14 (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. For example, consider the two element set $\mathbb{B} := \{0, 1\}$, and let $T : \{\star\} \nrightarrow \mathbb{B} := \{(\star, 0), (\star, 1)\} \simeq \{0, 1\}$. Consider the composite of T with the copy relation:

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

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

Remark 1.6.15. 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 partial functions; in other words, this diagrammatic equation expresses *determinism*.

Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 1.6.14 and Remark 1.6.15, 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 the commutative comonoid equations, 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 is only a commutative comonoid. In fact, quantum physicists *do* do this; see Dodo: [CK17].

1.6.2 What is an update?

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

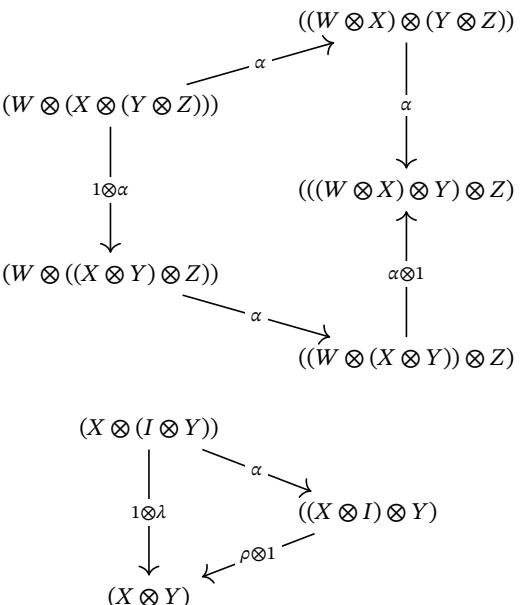
Perhaps the most familiar setting for an update is a database. In a database, an *entry* often takes the form of pairs of *fields* and *values*. For example, where a database contains information about employees, a typical entry might look like:

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

There are all kinds of reasons one might wish to update the value of a field: Jono might legally change their name, a year might pass and Jono's age must be incremented, Jono might be promoted or demoted or get a raise and so on. It was the concern of database theorists to formalise and axiomatise the notion of updating the value of a field independently of the specific programming language implementation of a database. The problem is reducible to axiomatising a *rewrite*: we can think of updating a value as first calculating the new value, then *putting* the new value in place of the old. Since often the new value depends in some way on the old value, we also need a procedure to *get* the current value. That was a flash-prehistory of *bidirectional transformations* [CFH⁺09], which then met applied category theory in e.g. [Gib12]. Following the monoidal generalisation of lenses in [WHBW21, HWW20], a rewrite as we have described above is specified by system diagrammatic equations in the margin, each of which we introduce in prose.

Definition 1.6.6 (Monoidal Category). A monoidal category consists of a category \mathcal{C} , a functor $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$, a monoidal unit object $I \in \text{Ob}(\mathcal{C})$, and the following natural isomorphisms – i.e. natural transformations with inverses, where multiple bar notation indicates variable object argument positions: an associator $\alpha : ((-\otimes=)\otimes\equiv) \mapsto (-\otimes(=\otimes\equiv))$, a right unit $\rho : \otimes I \mapsto -$, and a left unit $\lambda : I \otimes - \mapsto -$. These natural isomorphisms must in addition satisfy certain *coherence* diagrams, to be displayed shortly.

Theorem 1.6.7 (Coherence for monoidal categories). The following pentagon and triangle diagrams are conditions in the definition of a monoidal category. When they hold, all composites of associators and unitors (and their inverses) are isomorphisms [Mac63, Ben64].



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



Definition 1.6.16 (Symmetric Monoidal Category). A symmetric monoidal category is a monoidal category with an additional natural isomorphism

$$\theta : - \otimes - \mapsto - \otimes -$$

Which satisfies the following pair of hexagons.

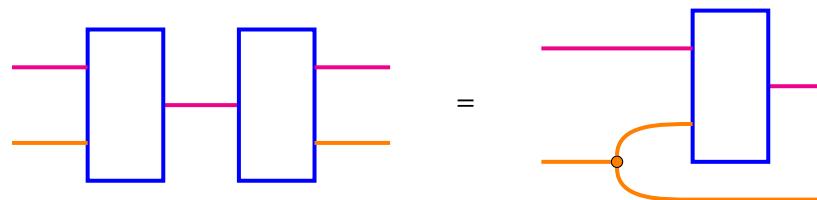
$$\begin{array}{ccc} (X \otimes (Y \otimes Z)) & \xrightarrow{\theta} & (Z \otimes (X \otimes Y)) \\ \alpha^{-1} \downarrow & & \downarrow \alpha \\ (X \otimes (Z \otimes Y)) & & ((Z \otimes X) \otimes Y) \\ 1 \otimes \theta \downarrow & & \downarrow \theta^{-1} \\ (X \otimes (Z \otimes Y)) & \xrightarrow{\alpha} & ((X \otimes Z) \otimes Y) \end{array}$$

$$\begin{array}{ccc} (X \otimes (Y \otimes Z)) & \xrightarrow{\theta} & ((Y \otimes Z) \otimes X) \\ \alpha \downarrow & & \downarrow \alpha^{-1} \\ ((X \otimes Y) \otimes Z) & & (Y \otimes (Z \otimes X)) \\ \theta^{-1} \downarrow & & \downarrow 1 \otimes \theta \\ ((Y \otimes X) \otimes Z) & \xrightarrow{\alpha^{-1}} & (Z \otimes (X \otimes Y)) \end{array}$$

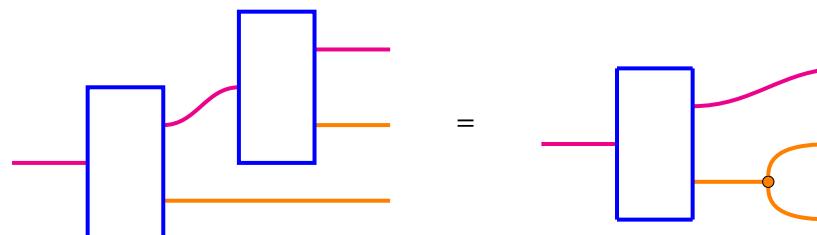
GetPut: Getting a value from a field and putting it back in is the same as not doing anything.



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



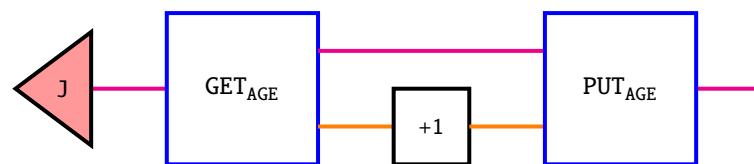
GetGet: Getting a value from a field twice is the same as getting the value once and copying it.



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

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

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



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, 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 (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 (the practical kind, not the one with crazy infinities), 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 $\{\star\} \rightarrow X$. 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.

Coherence is about getting rid of syntactic bureaucracy. Addition for example is a commutative monoid, which satisfies equations that let us forget about bracketings, zeroes, and the ordering of summation.

$$(x + (y + z)) = ((x + y) + z)$$

$$x + 0 = x = 0 + x$$

$$x + y = y + x$$

Now the situation is that we have replaced the associativity equation with associator natural transformations, unit equations with unitors (and commutation with natural isomorphisms θ that are depicted as twisting wires in string diagrams for symmetric monoidal categories.)

$$((X \otimes Y) \otimes Z) \xrightarrow{\alpha_{XYZ}} (X \otimes (Y \otimes Z))$$

$$(X \otimes I) \xrightarrow{\rho_X} X \xrightarrow{\lambda_X} (I \otimes X)$$

Recalling that we're happy with isomorphism in place of equality, coherence conditions ask that every possible composite of these structural operations is an isomorphism. In terms of the graphical language for monoidal categories, this means that string diagrams up-to-(processive)-planar-isotopy (and connectivity of wires in the case of symmetric monoidal categories) represent equivalent-up-to-isomorphism morphisms in an appropriate monoidal category. The reader is referred to [Sel10, JS91, Joy] for details.

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

The following rebuttal draws on Harold Abelson's introductory lecture to computer science [har19] (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.

NOW WE DEFINE A VERY USEFUL STRUCTURE. Spiders – or special frobenius algebras (introduced in [Paq08]) – will be important throughout this thesis.

Definition 1.6.17 (Spiders). I will describe the behaviour of a spider as a PROP [nLac], which as far as we care is just a way to list out processes and equations that their composites satisfy in a symmetric monoidal category. We say that an object is *equipped* with a spider when it has the following processes:



That satisfy the following equations:

associativity	coassociativity	commutativity	cocommutativity

unitality	special	counitality	Frobenius

Spiders imply strong compact closure; self-dual cups and caps may be viewed as composite spiders. These are the cups which we use in pregroup diagrams to come. The equations for cups and caps are derived from spider rules, taking advantage of equality up to connectivity of wires.

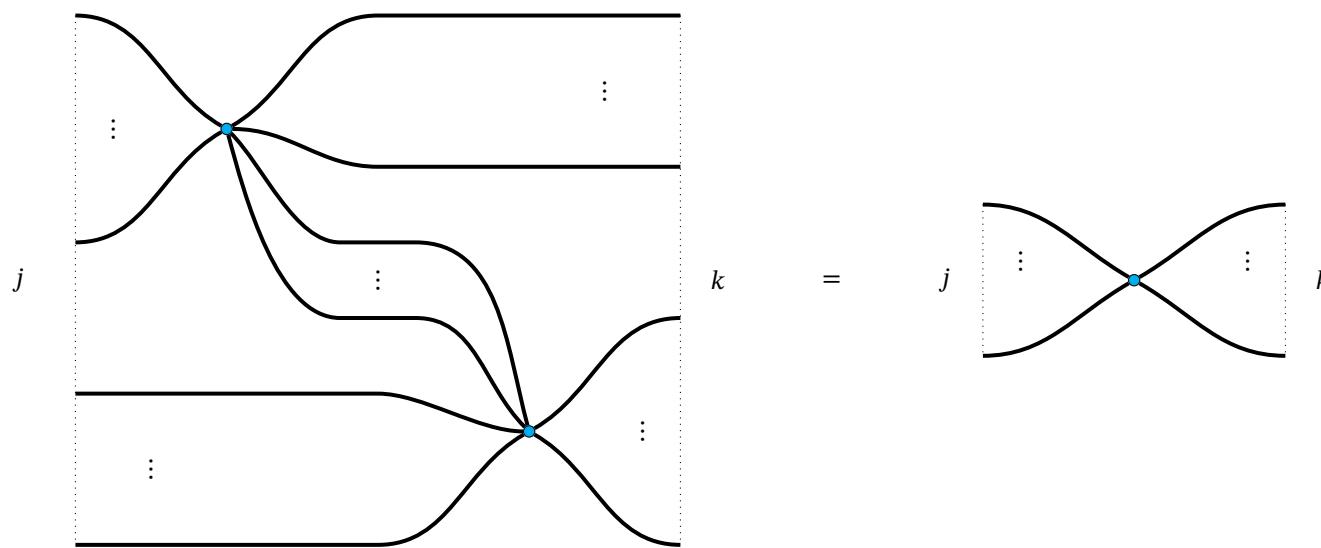
(Connectivity)	(Commutativity)	(Frobenius)	(Unit)	(Counit)
=	=	=	=	=

Terminology 1.6.18. A *hypergraph category* [FS18] is a symmetric monoidal category in which every object is equipped with a choice of spider, and spiders of tensors are tensors of spiders, as illustrated below:

The first diagram shows two configurations of wires labeled $(X \otimes Y)$. The left configuration has two wires, one labeled X and one labeled Y , meeting at a blue dot. The right configuration has a single wire labeled X with a blue dot, representing a spider. An equals sign indicates they are equivalent. The second diagram shows three configurations of wires labeled $(X \otimes Y)$. The left configuration has a wire labeled $(X \otimes Y)$ with a blue dot. The middle configuration has two wires, one labeled $(X \otimes Y)$ and one labeled $(X \otimes Y)$, meeting at a blue dot. The right configuration has a single wire labeled $(X \otimes Y)$ with a blue dot. An equals sign indicates they are equivalent. The third diagram shows four configurations of wires labeled X and Y . The top-left and bottom-right configurations have two wires labeled X meeting at a blue dot. The top-right and bottom-left configurations have two wires labeled Y meeting at a blue dot. The top-right and bottom-left configurations are swapped relative to the top-left and bottom-right configurations. An equals sign indicates they are equivalent.

Example 1.6.19. The category of sets and relations with cartesian product as \otimes is a hypergraph category, as is the category \mathbf{FdVect}^\otimes , the category of finite dimensional vector spaces and linear maps with the Kronecker product as \otimes .

Remark 1.6.20 (Spiders are easy). All connected configurations of spiders with the same number of inputs and outputs are equal. Intuitively, whereas wires connect end-to-end in string diagrams, spiders give us *multiwires* that we may freely split and connect.



Remark 1.6.21 (Only connectivity matters). In hypergraph categories, by recovery of strong compact closure, we may freely reason without a convention for reading direction of processes. By the multiwire property, all diagrams with the same connectivity are equal. Hence, *only connectivity matters*.

1.7 Previously, on DisCoCat

DisCoCat is a research programme in applied mathematical linguistics that is **Distributional**, **Compositional** and **Categorical**. In this section I will recount a selective development of DisCoCat as relevant for this thesis.

1.7.1 Lambek's Linguistics

Jim Lambek was a jovial man who always carried a wad of twenties. I can't do better than Moortgat's history and exposition of typological grammar in [Moo14], so I will borrow Moortgat's phrasing and summarise Lambek's role in the story. Typological grammar originated in two seminal papers by Lambek in 1958 [Lam58] and 1961 [Lam61], where Lambek sought "to obtain an effective rule (or algorithm) for distinguishing sentences from non-sentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages [...]" . The method is to assign grammatical categories – parts of speech such as nouns and verbs – logical formulae. Whether a sentence is grammatical or not is obtained from deduction using these logical formulae in a Gentzen-style sequent proof.

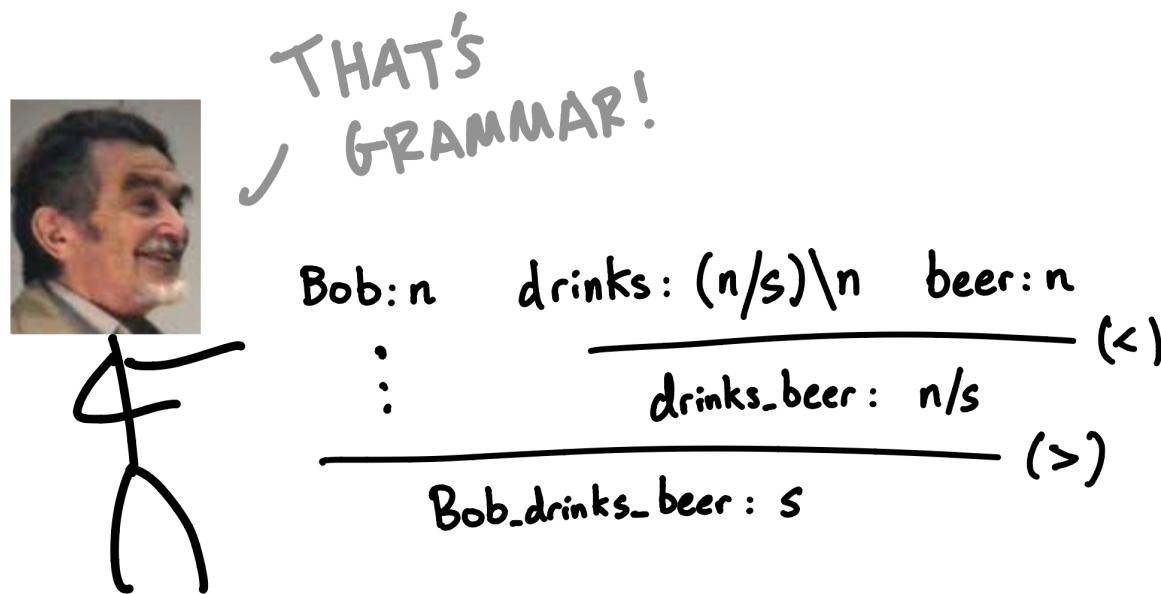


Figure 1.17: In English, we may consider a noun to have type n , and a transitive verb (n/s) \ n , to yield a well-formedness proof of Bob drinks beer. The type formation rules for such a grammar are intuitive. Apart from a stock of basic types \mathbb{B} that contains special final types to indicate sentences, we have two type formation operators ($- / =$) and ($- \backslash =$), which along with their elimination rules establish a requirement that grammatical categories require other grammatical categories to their left or right. This is the essence of Lambek's calculi NL and L. CCGs keep the same minimal type-formations, but include extra sequent rules such as type-raising and cross-composition.

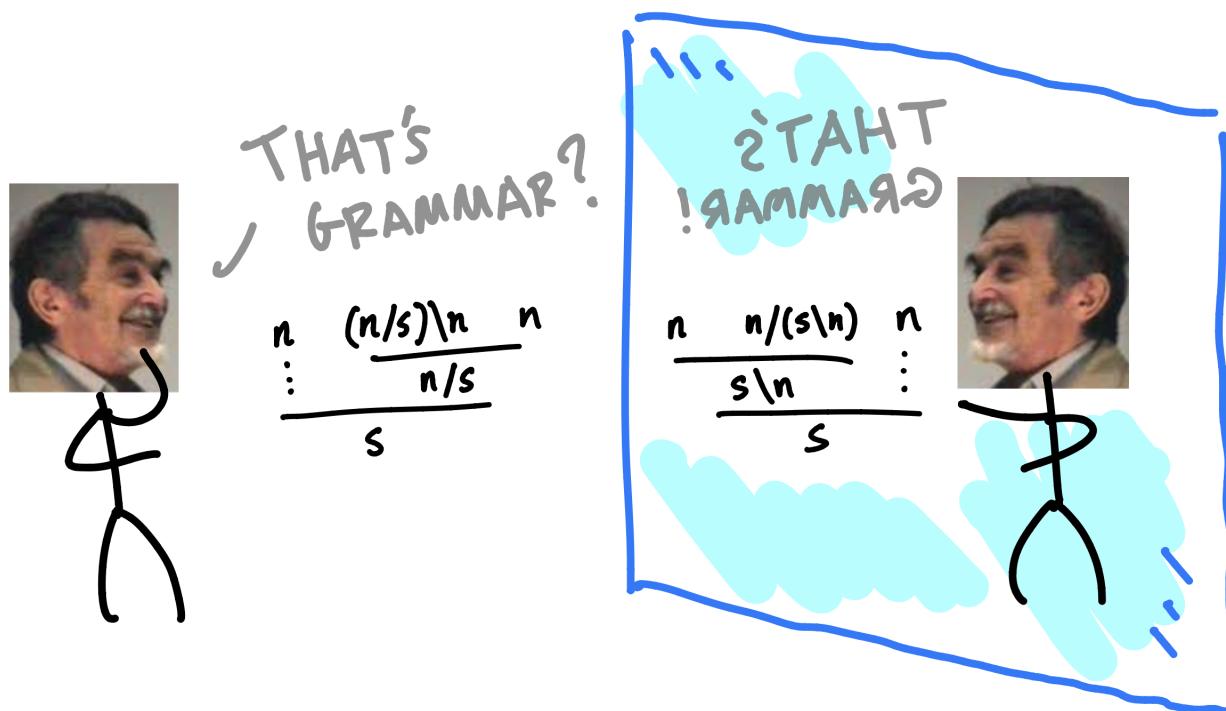
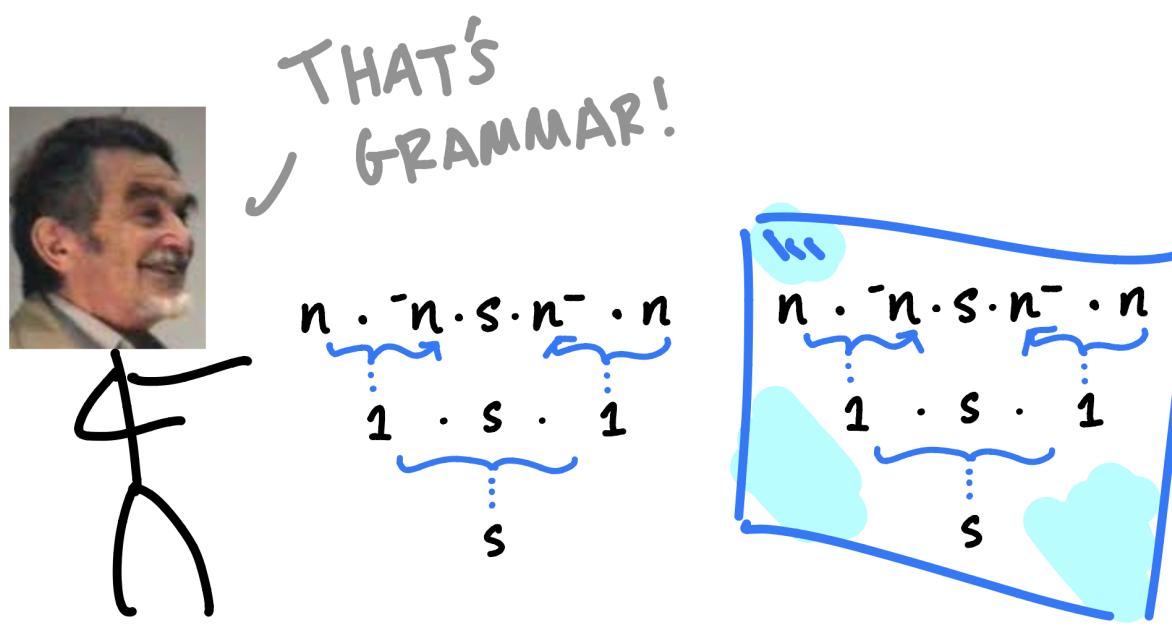


Figure 1.18: We can notice an asymmetry in the above formulation when we examine the transitive verb type $(n/s) \setminus n$ again; it asks first for a noun to the right, and then a noun to the left. We could just as well have asked for the nouns in the other order with the typing $(n/s) \setminus n$ and obtained all of the same proofs.



$$n \cdot (-^1n \cdot s \cdot n^{-1}) \cdot n \rightarrow (n \cdot -^1n) \cdot s \cdot (n^{-1} \cdot n) \quad (1.1)$$

$$\rightarrow 1 \cdot s \cdot 1 \quad (1.2)$$

$$\rightarrow s \quad (1.3)$$

Figure 1.19: To eliminate this asymmetry, Lambek devised pregroup grammars. Whereas a group is a monoid with inverses up to left- and right-multiplication, a pregroup weakens the requirement for inverses so that all elements have distinct left- and right- inverses, denoted x^{-1} and $-^1x$ respectively. Eliminating or introducing inverses is a non-identity relation on elements of the pregroup, so we have axioms of the form e.g. $x \cdot -^1x \rightarrow 1 \rightarrow -^1x \cdot x$. In this formulation, denoting the multiplication with a dot, both $(n/s) \setminus n$ and $(n/s) \setminus n$ become $-^1n \cdot s \cdot n^{-1}$, which just wants a noun to the left and a noun to the right in whatever order to eliminate the flanking inverses to reveal the embedded sentence type. Now we can obtain the same proof of correctness as a series of algebraic reductions.

1.7.2 Coecke's Composition

Figure 1.20: Meanwhile, an underground grunge vagabond moonlighting as a quantum physicist moonlighting as a computer scientist was causing a shortage of cigars and whiskey in a small English town. He noticed a funny thing about the composition of multiple non-destructive measurements of a quantum system, which was that information could be carried, or flow, between them. So he wrote a paper [Coe04], which contained informal diagrams that looked like this.

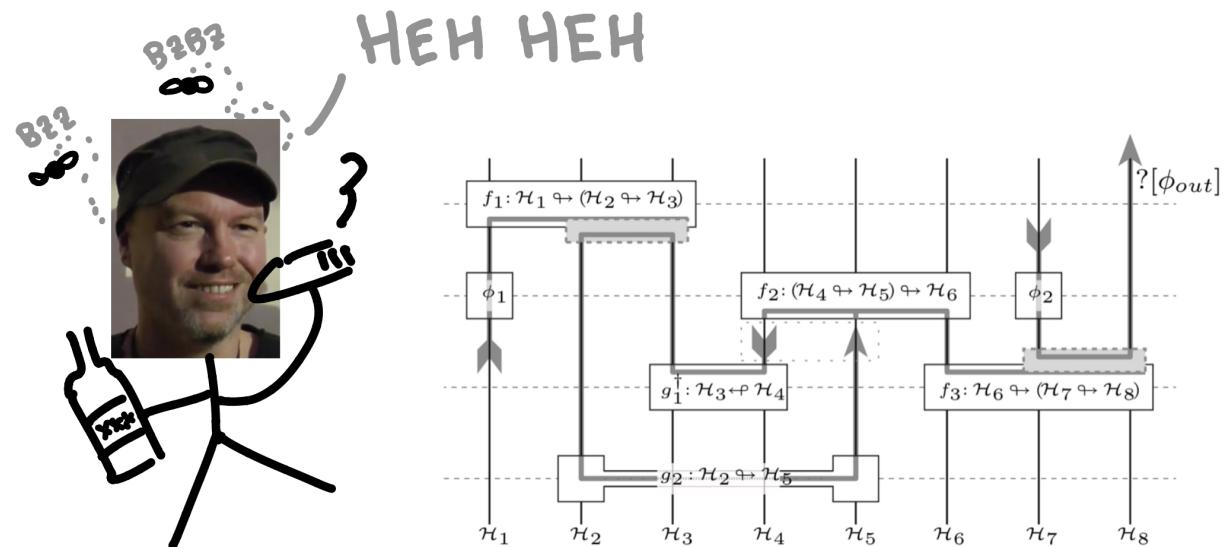
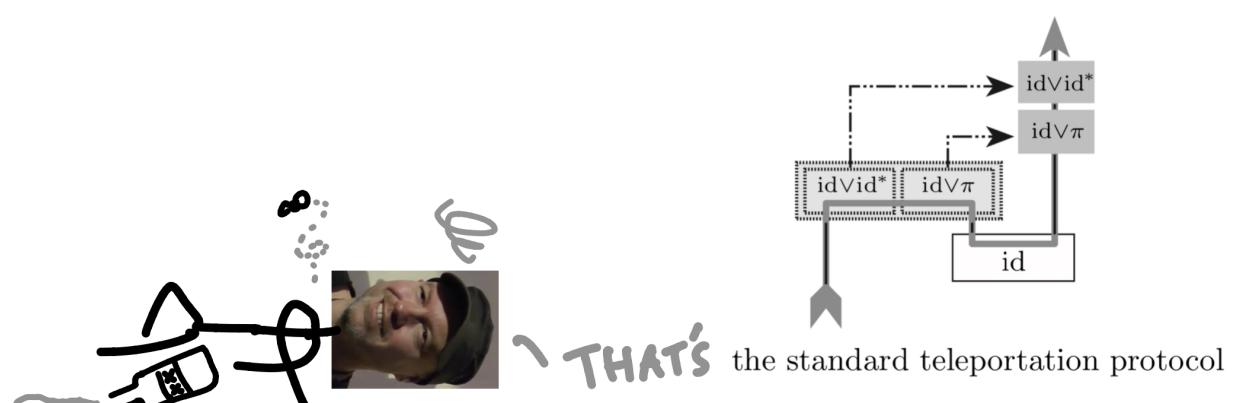


Figure 1.21: There were two impressive things about these diagrams. First, the effects such as transparencies for text boxes and curved serifs for angled arrows give a modern feel, but they were done manually in MacDraw, the diagrammatic equivalent of sticks and stones. Second, though the diagrams were informal, they provided a way to visualise and reason about entanglement that was impossible by staring at the equivalent matrix formulation of the same composite operator. The most important diagram for our story was this one, which captures the information flow of quantum teleportation.



1.7.3 Categorical quantum mechanics

Figure 1.22: Category theorists and physicists such as Abramsky and Baez were excited about these diagrams, which looked like string diagrams waiting to be made formal. The graphical cups and caps in the important diagram were determined to correspond to a special form of symmetric monoidal closed category called strong compact closed [Abr09].

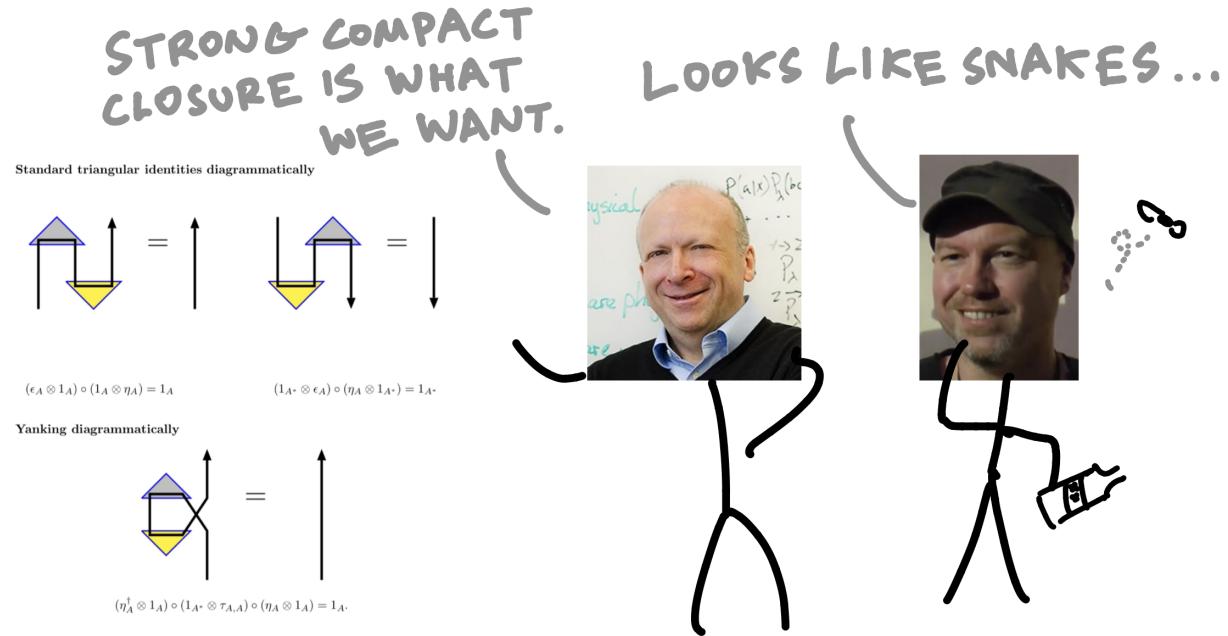


Figure 1.23: Diagrammatically, reasoning in a strongly compact closed category amounts to ignoring the usual requirement of processiveness and forgetting the distinction between inputs and outputs, so that "future" outputs could curl back and be "past" inputs. This formulation also gave insight into the structure of quantum mechanics. For example, the process-state duality of strong compact closure manifested as the Choi–Jamiołkowski isomorphism [Cho75, Jam72].



Figure 1.24: However, dealing with superpositions necessitated using summation operators within diagrams, which is cumbersome to write especially when dealing with even theoretically simple Bell states. An elegant diagrammatic simplification arose with the observation that special- \dagger -frobenius algebras [CPP09], or spiders, correspond to choices of orthonormal bases [CPV13] in **FdHilb**, the ambient setting of finite-dimensional hilbert spaces. Not only did this remove the need for summation operators, it also revealed that strong compact closure was a derived, rather than fundamental structure, since spiders induce compact closed structure.

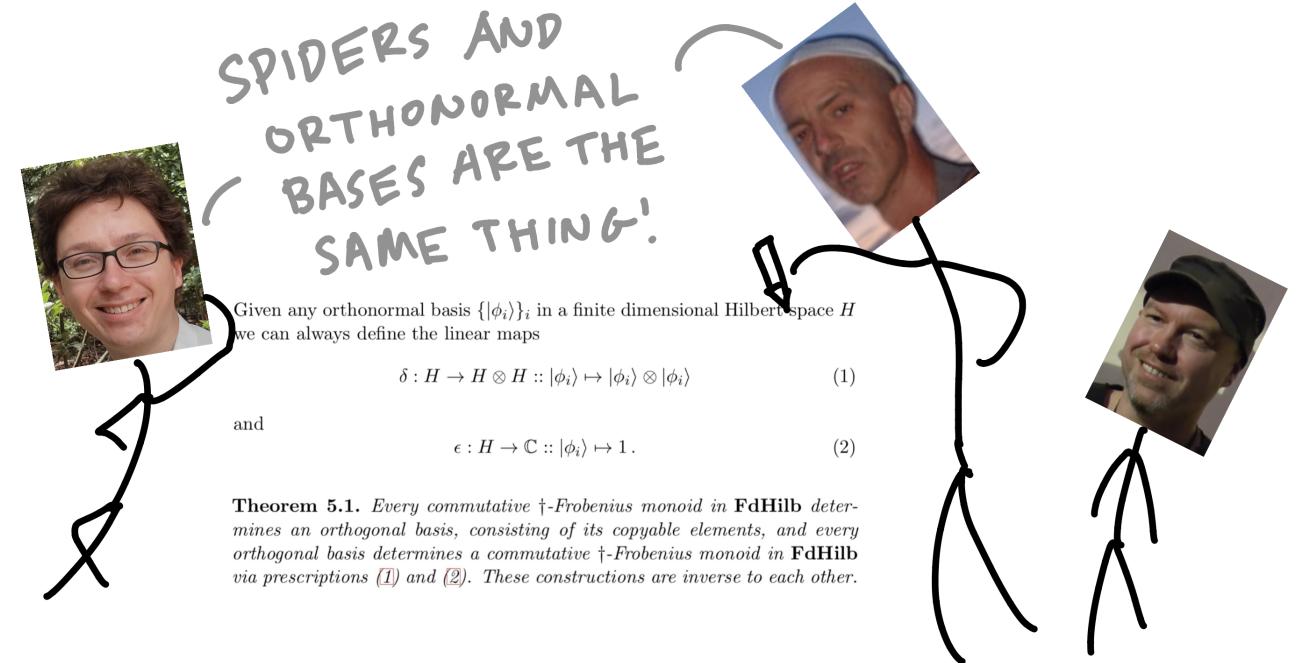
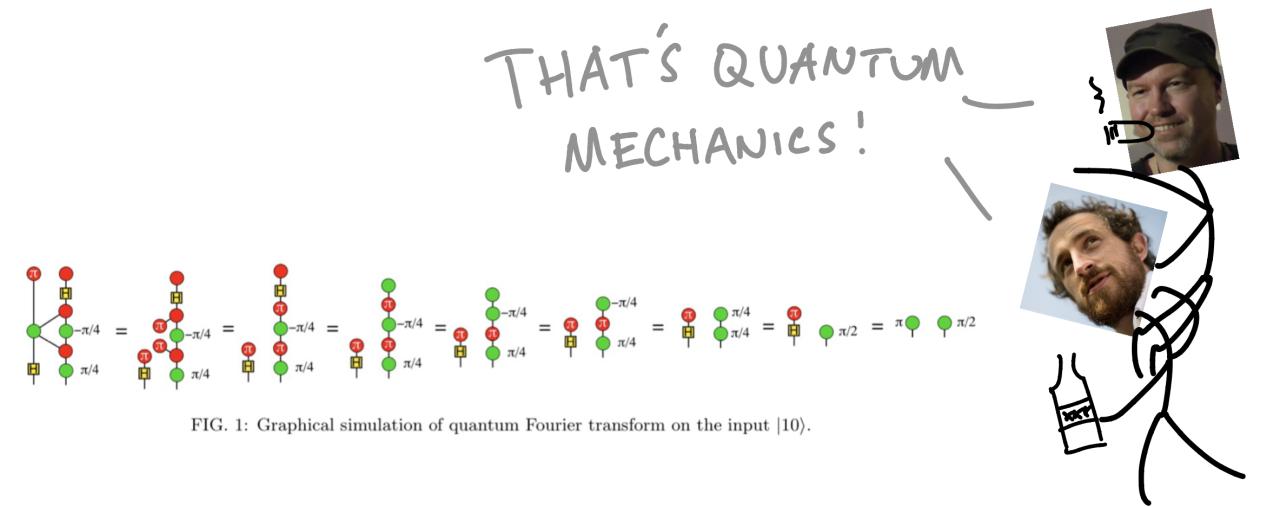


Figure 1.25: And so the stage was set for a purely diagrammatic treatment of ZX quantum mechanics via interacting spiders [CD11]. The story of ZX diverges away from our interest, so I will summarise what happened afterwards. In no particular order, the development of ZX went on to accommodate a third axis of measurement [CE11, dZ14] to yield a ZXW calculus then proven to be complete for the usual hilbert space formalism for quantum [Had17, Ng18, PWS⁺23]. There are at the time of writing two expository books [CK17, CG23], and ZX-variants are becoming an industry standard for quantum circuit specification and rewriting.



1.7.4 Enter computational linguistics

Figure 1.26: Somewhere in Canada at the turn of the millennium, Bob met Jim, who saw something familiar about the diagram for quantum teleportation. The snake equation for compact closure looked a lot like the categorified version of introducing and eliminating pregroup types.

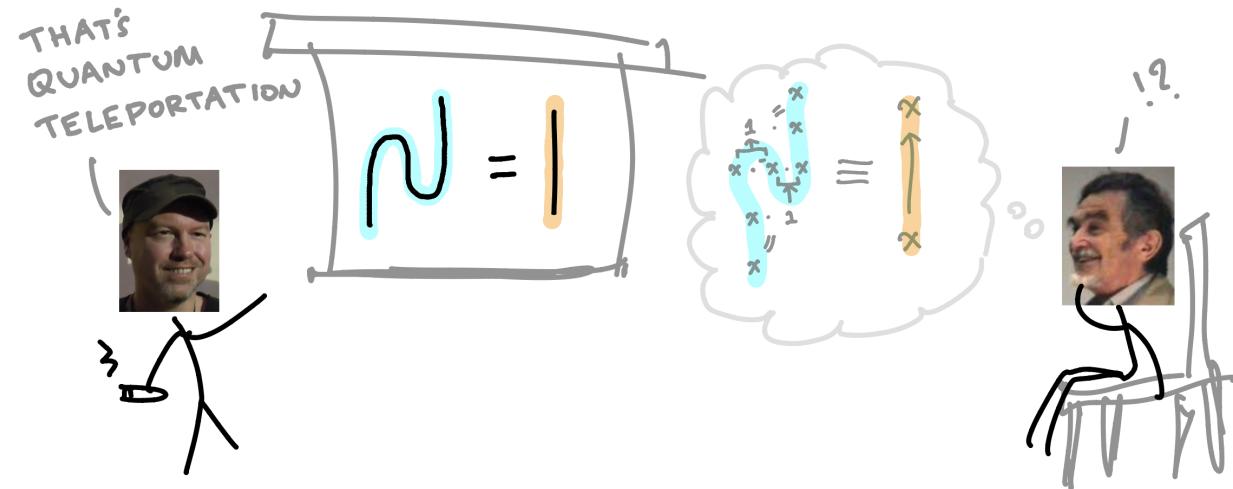


Figure 1.27: Bob and Jim's meeting put the adjectives *compositional* and *categorical* on the same table, but the cake wasn't ready. Two more actors Steve and Mehrnoosh were required to introduce *distributional*, which refers to Firth's maxim "you shall know a word by the company it keeps" [Fir57]. In its modern incarnation, this refers generally to vector-based semantics for words, where it is desirable but not necessarily so (as in the case of generic latent space embeddings by an autoencoder) that proximity of vectors models semantic closeness.

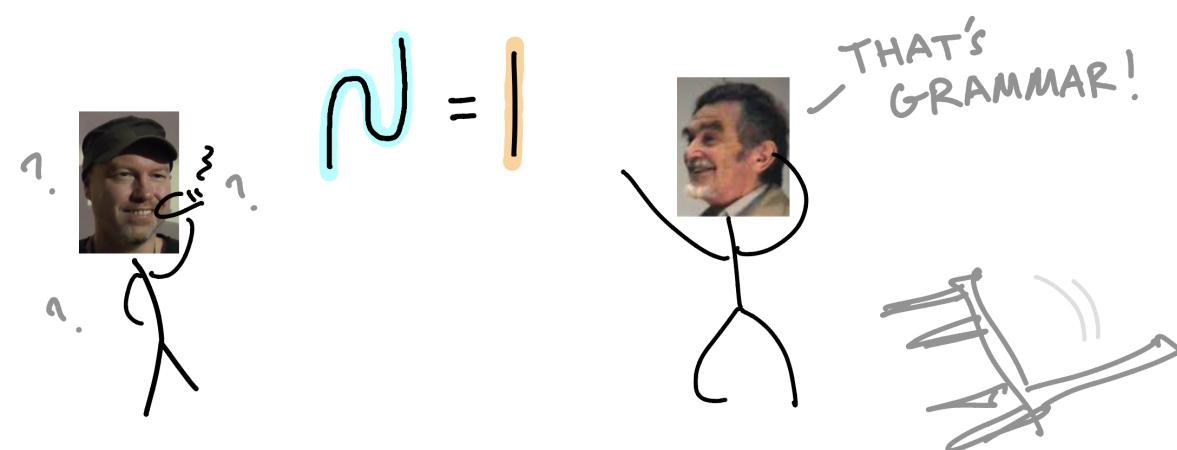


Figure 1.28: Steve Clark was a professor in the computer science department at Oxford, and he was wondering how to compose vector-based semantic representations. Steve asked Bob, who realised suddenly what Jim was talking about. Mediated by the linguistic expertise of Mehrnoosh who was a postdoctoral researcher in Oxford at the time, pregroup diagrams were born. The basic types n and s are assigned finite-dimensional vector spaces, concatenation of types the Kronecker product \otimes , and by the isomorphism of dual spaces in finite dimensions there is no need to keep track of the left- and right- inverse data. Words become vectors, and pregroup reductions become bell-states, or bell-measurements, depending on whether one reads top-down or bottom-up. There was simply no other game in town for an approach to computational linguistics that combined linguistic compositionality with distributional representations [CSC10].



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

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

Figure 1.29: In [SCC13, SCC16], the trio realised that spiders could play the role of relative pronouns, which was genuinely novel linguistics. If one follows the noun-wire of "movies", one sees that by declaring the relative pronoun to be a vector made up of a particular bunch of spiders-as-multiwires, "movies" is copied to be related to the "liked" word, copied again by "which" to be related to the "is-famous" word, and a third time to act as the noun in the whole noun-phrase. This discovery clarified a value proposition: insights from quantum theory could be applied in the linguistic setting: for example, density matrices were used to model semantic ambiguity [ML20].

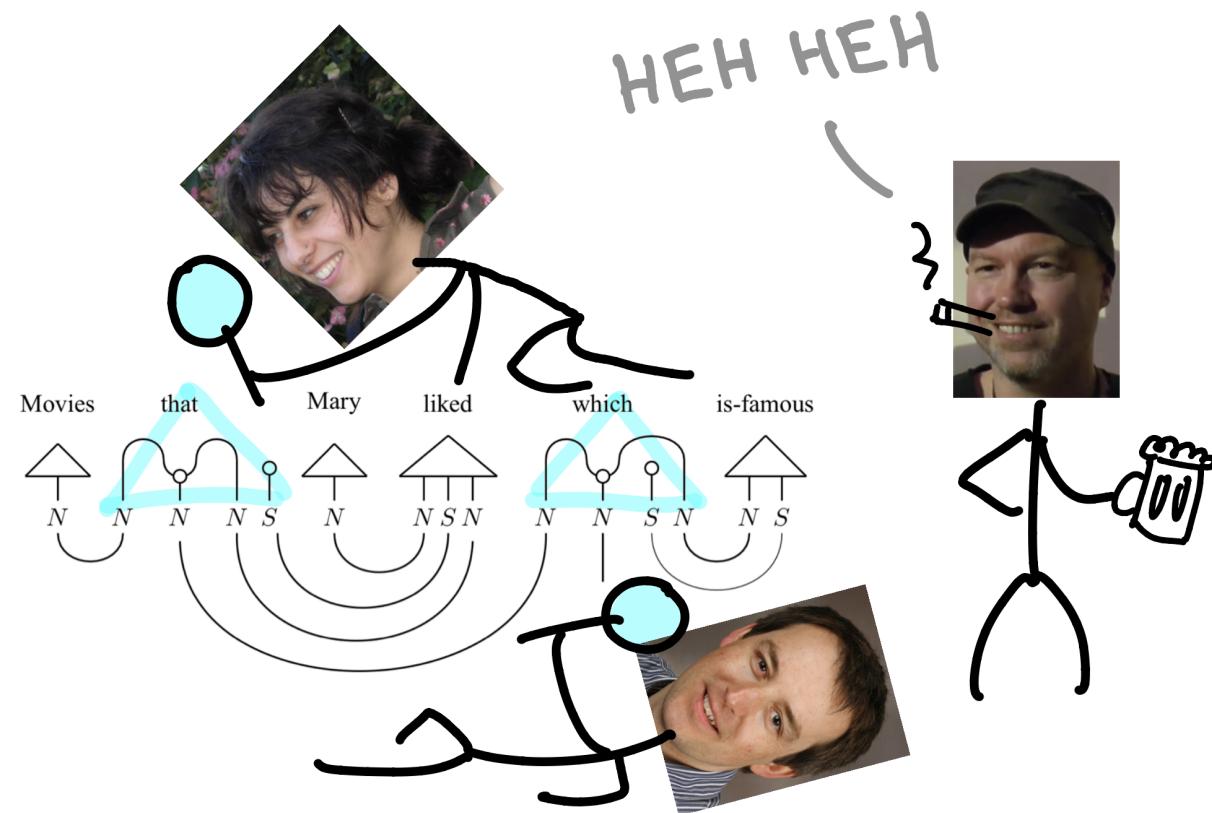


Figure 1.30: Keeping the structure of the diagrams but seeking set-relational rather than vector-based semantics, a bridge was made between linguistics and cognitive science in *Interacting Conceptual Spaces I* [BCG⁺17]. Briefly, Gärdenfors posits [Gär14] that spatial representations of concepts mediate raw sense data and symbolic representations – e.g. red is a region in colourspace – and moreover that concepts ought to be spatially convex – e.g. mixing any two shades of red still gives red. A new point in the value proposition arose: that new mathematics would arise from investigating the linguistic-quantum bridge, e.g. generalised relations [MG17]. Although labelled as if it is the first in a series, the paper never saw a sequel by the same title, blocked by an apparently simple but actually tricky theoretical problem: while this convex-relational story worked for conceptual adjectives modifying a single noun (such as for "sweet yellow bananas"), there was difficulty in extending the story to work for multiple objects interacting in the same space, as in "cup on table in room". It couldn't be worked out what structure a sentence-wire in **ConvexRel** ought to have in order to accommodate (in principle) arbitrarily many objects and spatial relations between them.

Definition 4. We define the category **ConvexRel** as having convex algebras as objects and convex relations as morphisms, with composition and identities as for ordinary binary relations.

Given a pair of convex algebras (A, α) and (B, β) we can form a new convex algebra on the cartesian product $A \times B$, denoted $(A, \alpha) \otimes (B, \beta)$, with mixing operation:

$$\sum_i p_i |(a_i, b_i)\rangle \mapsto \left(\sum_i p_i a_i, \sum_i p_i b_i \right)$$

This induces a symmetric monoidal structure on **ConvexRel**. In fact, the category **ConvexRel** has the necessary categorical structure for categorical compositional semantics:

Theorem 1. The category **ConvexRel** is a compact closed category. The symmetric monoidal structure is given by the unit and monoidal product outlined above. The caps for an object (A, α) are given by:

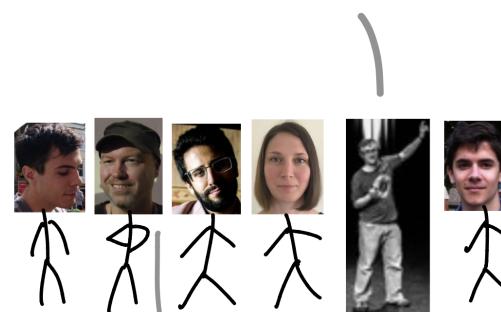
$$\circlearrowleft : I \rightarrow (A, \alpha) \otimes (A, \alpha) :: \{(*, (a, a)) \mid a \in A\}$$

the cups by:

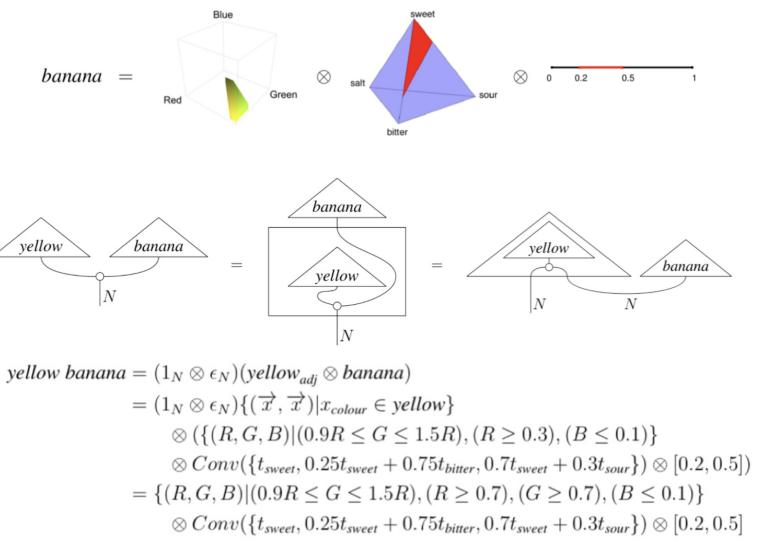
$$\circlearrowright : (A, \alpha) \otimes (A, \alpha) \rightarrow I :: \{((a, a), *) \mid a \in A\}$$

and more generally, the multi-wires by:

$$\dots : A \otimes \dots \otimes A \rightarrow A \otimes \dots \otimes A :: \{ ((a, \dots, a), (a, \dots, a)) \mid a \in A \}$$



BUT WHERE CAN WE FIND A SENTENCE-SPACE
BIG ENOUGH FOR SPATIAL RELATIONS ON MANY THINGS?



1.7.5 I killed DisCoCat, and I would do it again.

DisCoCat then diverges from the story I want to tell. In no particular order, QNLP was done on an actual quantum computer [LPM⁺23], some software packages were written [kar23], and some art happened [A q]. It is a common evolutionary step in linguistics that theories "break the sentential barrier", moving from sentence-restricted to text- or discourse-level analysis. The same thing happened with DisCoCirc, due to a combination of practical constraints and theoretical ambition.

Figure 1.31: On the practical side, wide tensors were (and remain) prohibitively expensive to simulate classically and actual quantum computers did not (and still do not) have many qubits, hence in practice pregroup diagrams were reduced to thinner and deeper circuits, often with the help of an additional simplifying assumption that sentence wires were pairs of noun wires in the illustrated form on the left. Theoretically, seeking dynamic epistemic logic, Bob had an epiphanous hangover (really) where he envisioned that these "Cartesian verbs" could be used in service of compositional text meanings, and he called this idea DisCoCirc [Coe20].



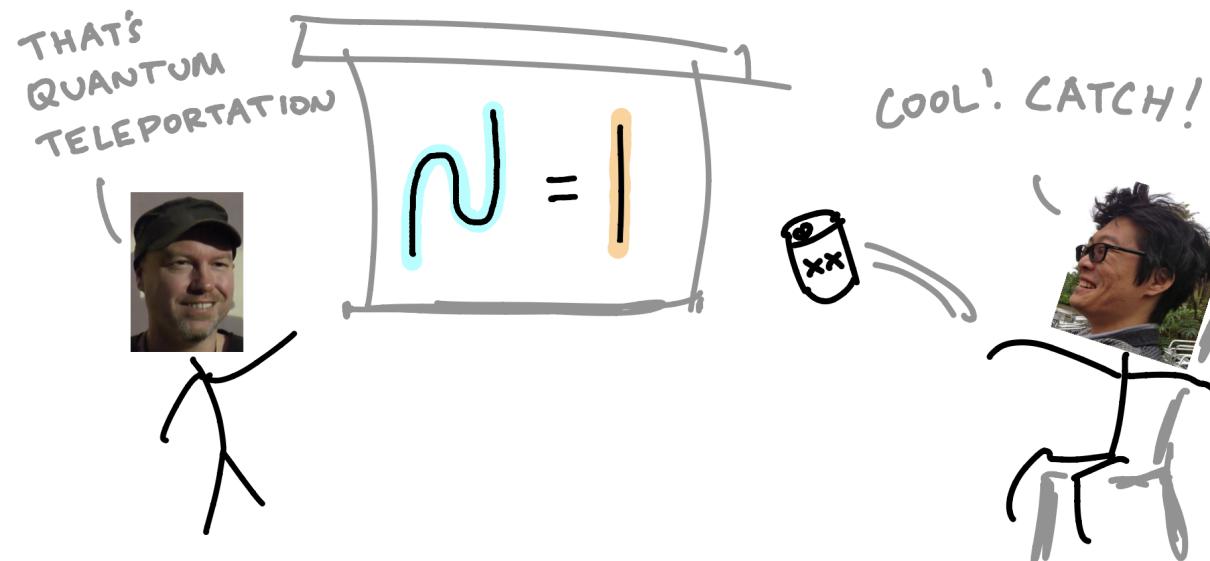


Figure 1.32: I met Bob in my master's in 2019, where he taught the picturing quantum processes course. When quantum teleportation was explained in half a minute by a diagram, I decided to pursue a DPhil in diagrammatic mathematics. In the last lecture, I threw Bob a cider, after which he seemed to like me.

I was shanghaied into thinking about diagrams for language. I was deeply dissatisfied with the content from the standpoint of my own intellectual integrity. Firstly, there seemed to me an unspoken claim that the presence of cups in pregroup diagrams (which implied a noncartesian and hence large tensor product) made it necessary to use quantum computers to effectively compute pregroup diagrams. I just could not believe that my brain required quantum computation to understand language. This implicit claim of kinship between quantum and linguistics was further entrenched by the analysis of the relative pronoun in terms of frobenius algebras, since spiders in \mathbf{Vect}^\otimes were the *sine qua non* of categorical quantum mechanics. The best steelman for spiders I have is that frobenius algebras (which are central to ubiquitous bicategories of relations [CW87, CKWW07]) just happen to be a ubiquitous mathematical structure that are well-suited to express the mathematics of connections, both in language and in quantum.

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

sitional, and distributional semantics; I just didn't believe that the role of quantum – however helpful or interesting – was *necessary*.

My first unsatisfactory attempt to extricate quantum from language was in my Master's thesis [Wan19]. It had been known for a while that a free autonomous category construction by Delpeuch [Del20] could potentially eliminate some of the cups in pregroup diagrams, yielding what amounted to a method to transform a pregroup diagram into a monoidal string diagram in the shape of a context-free grammar tree. This trick had the limitation that freely adding directed cups and caps to a string diagrammatic signature did not turn a symmetric monoidal category into a (weakly) compact closed one, rather just into a monoidal category where the original wires had braidings, but all the new left and right dual wires did not; this presented difficulties in accounting for iterated duals for higher-order modifiers such as adverbs in grammatical types, and had nothing to say about spiders. I tried to generalise this trick to freely adding arbitrary diagrammatic gadgets to string diagrams, but my assessor Samson pointed out that it was nontrivial to determine whether such constructions were faithful. Nowadays there is a lot of categorical machinery lying around, and there are many different ways one could come up with ways to remove cups and caps. But I didn't learn about these techniques until later, and these approaches still wouldn't have addressed the issues that come with only having a sentence-wire.

Figure 1.33: Then COVID happened. During the first lockdown, I visited Bob's garden under technically legal circumstances, and I suggested a solution to the longstanding problem of representing linguistic spatial relationships. My theoretical concern was the culprit: the initial attempts at the problem failed because the approach was to find a single sentence object s in which one could paste the data of arbitrarily many distinct spatial entities. The simple solution was a change in perspective.

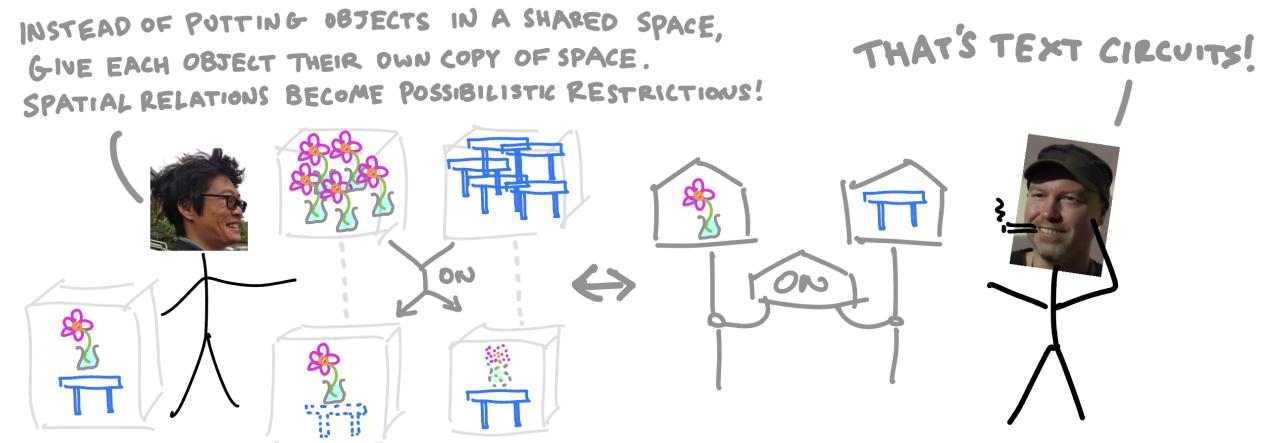


Figure 1.34: That this move of splitting up the sentence-wire into a sentence-dependent collection of wires was sufficient to solve what had appeared to be a difficult problem prompted some re-examination of foundations. The free autonomisation trick in conjunction with sentence-wire-as-tensored-nouns seemed promising, but it became clear that right way to drown a DisCoCat thoroughly was to explain and eliminate the spiders.

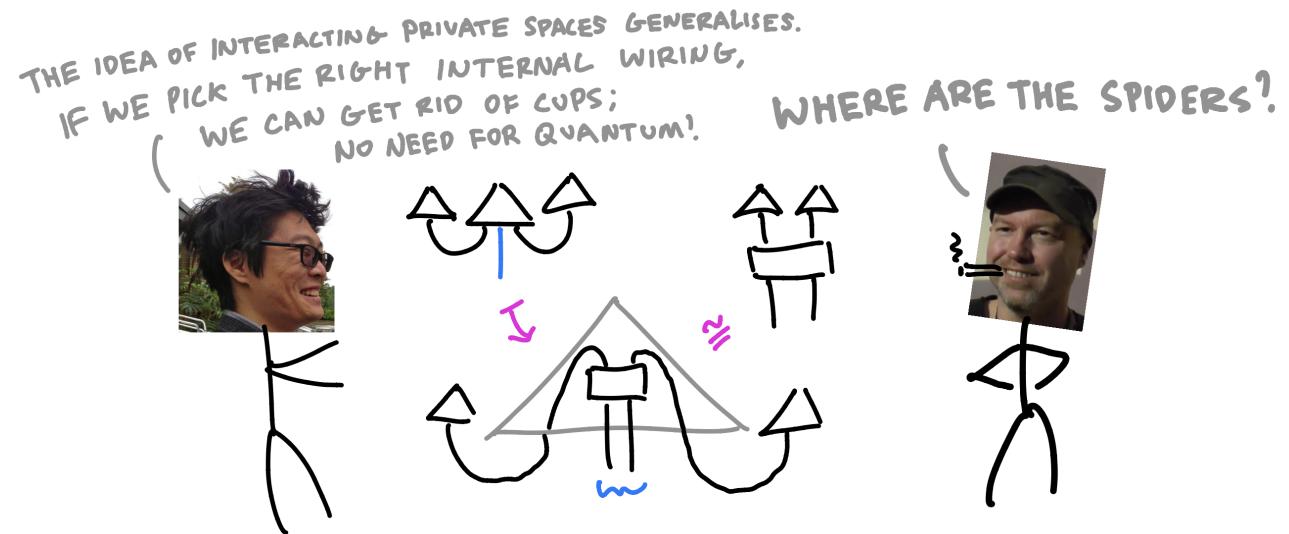


Figure 1.35: I then discovered that by interpreting spiders as the well-known "pair of pants" algebra in a compact closed monoidal setting allowed for a procedure in which the final form was purely symmetric monoidal – the absence of cups and caps meant that there was no practical necessity to interpret diagrams on quantum computers: *any* computer would suffice. The role of spiders for relative pronouns was illuminated in the presence of splitting the sentence wire: the pair-of-pants are the algebra of morphism composition, and splitting the sentence wire into a collection of nouns allowed relative-pronoun-spiders to pick out the participating nouns to compose relationships onto.

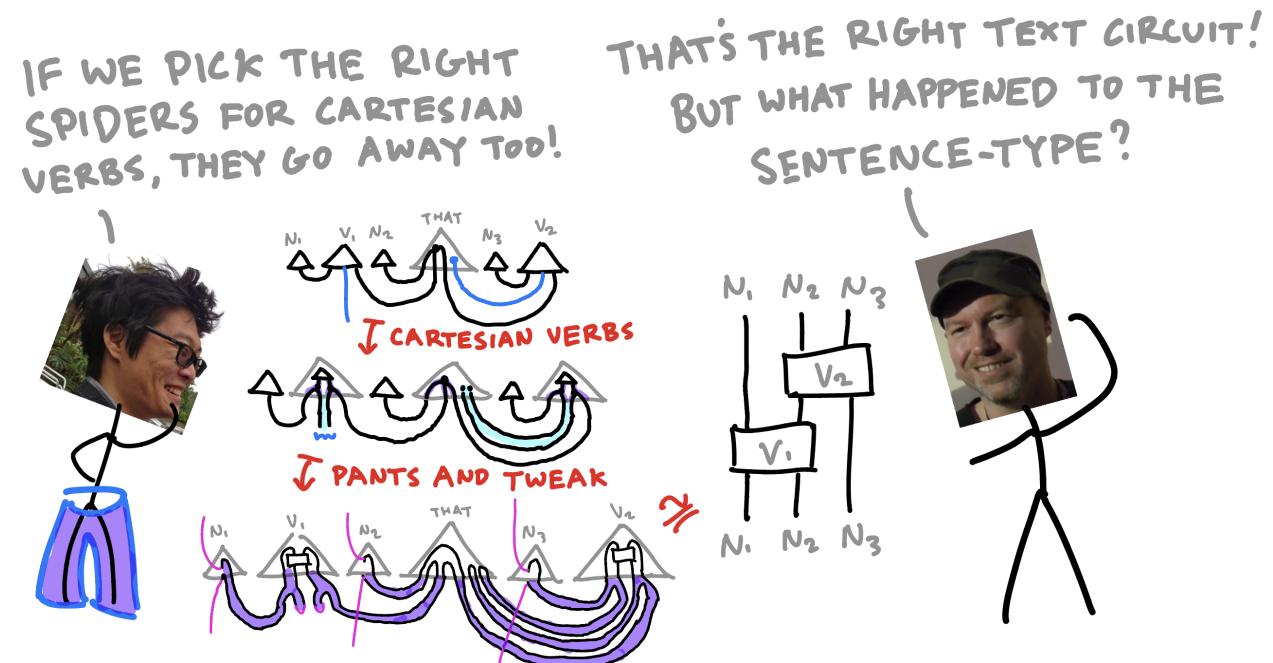
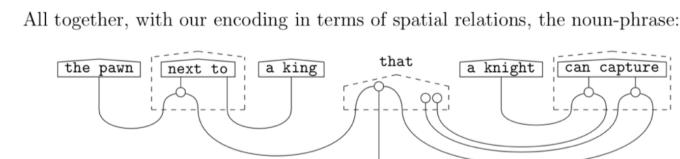


Figure 1.36: A coherent conservative generalisation of DisCoCat with less baggage had emerged, or rather, DisCoCirc was placed to formally subsume DisCoCat. It was now understood that the sentence type was a formal syntactic ansatz for the sake of grammar, which was to be interpreted in the semantic domain not as a single wire, but as a sentence-dependent collection of wires. It was further realised that the complexity of pregroup diagrams was due to grammar – the topological deformation of semantic connections to fit the one-dimensional line of language – whereas the essential connective content of language could be expressed in a simple form that distilled away the bureaucracy of syntax.

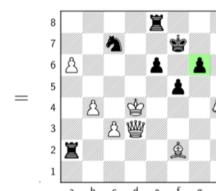
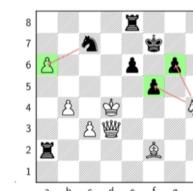


Figure 1.37: We wrote up the story about spaces in [WC21], the spiritual successor to *interacting conceptual spaces I*. We could formally calculate the meanings of sentences that used linguistic spatial relations, all using a simple and tactile diagrammatic calculus.

BETTER?
(



now yields the pawn we aimed to characterise, as now we obtain:

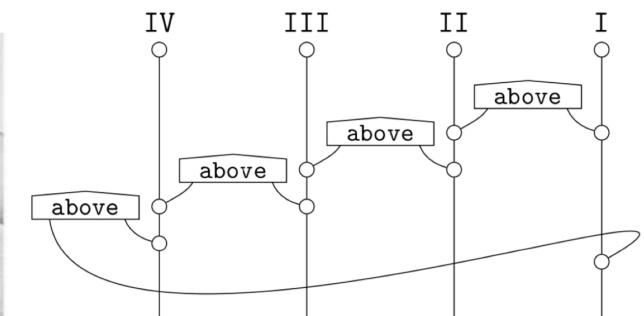


=



NO! SHOW A CIRCUIT!

DON'T START WITH A PATHOLOGICAL EXAMPLE, IDIOT!

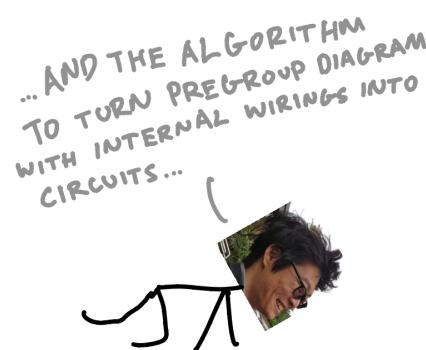


HEH HEH

Figure 1.38: The paper on spatial relations actually came very late, because I was busy with Bob's ludicrous request to go turn "all of language" into circuits. I bitched and moaned about how I wasn't a linguist and how it was an impossible task, but I was in too deep to back out.



Figure 1.39: I suppose the nice thing about aiming for the moon is that even failure might mean you leave orbit. So I settled for what I thought was a sensible fragment of English, for which I devised internal wirings and an algorithm that transformed pregroup diagrams with the internal wirings into circuit form. Many tiring diagrams later, I presented my results in the first draft of "distilling text into circuits". It was exhausting.

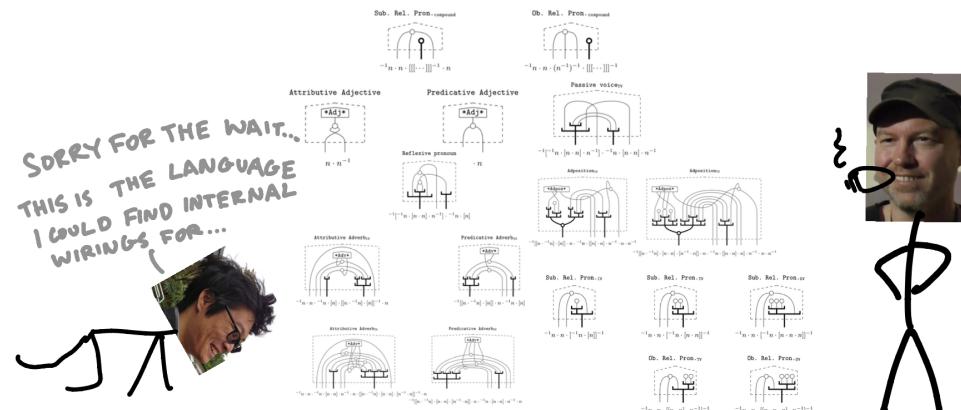


Algorithm 1: Grammar to Circuit Algorithm

```

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

```



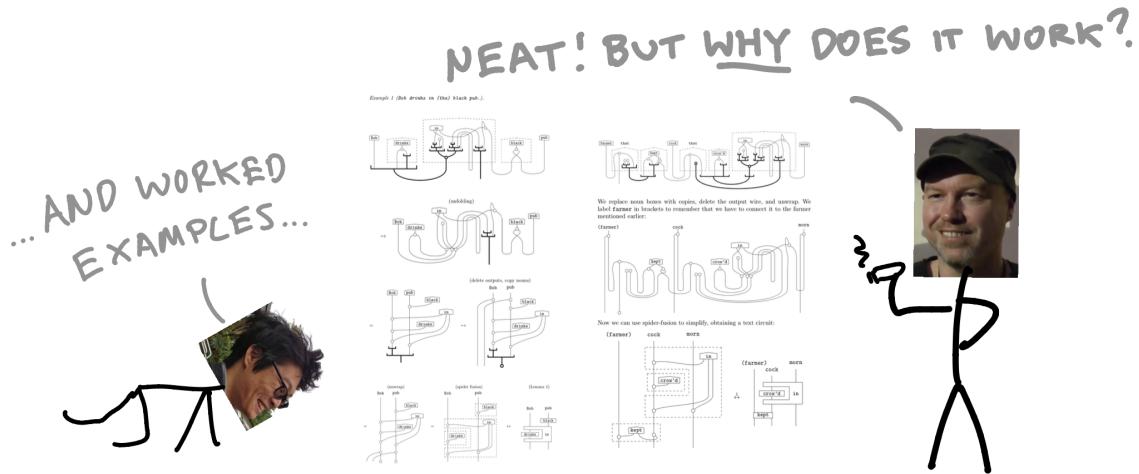


Figure 1.40: Bob had a good point. Everything worked, but we had no understanding as to why, and accordingly, whether or not it would all break. At this point in time, Jonathon Liu, a masters' student I tutored during COVID, had committed the grave error of thinking diagrams were cool, and was now hanging out with me and Bob. After understanding the procedure, Jono independently devised the same arcane internal wirings as I had, but neither of us could explain how we did it. So we had evidence of an underlying governing structure that was coherent but inarticulable.



Figure 1.41: I realised that our intuitions were coming from an implicit productive grammar, rather than a parsing one, and that the path of least resistance for obtaining formal guarantees for the language-to-circuit procedure was to just handcraft a generative grammar for the fragment of language we were interested in. This meant scrapping everything in the first draft and starting again from scratch. Bob always had a word of gentle encouragement, giving me the motivation to persevere. So now we had two ways to obtain text circuits.

One from pregroups (which Jono had extended the technique for to CCGs and dependency grammars in his master's thesis [Liu21]), and one from hand-crafted productive grammars. Then came time for me to write my thesis, and there were three salient questions I wanted to address.

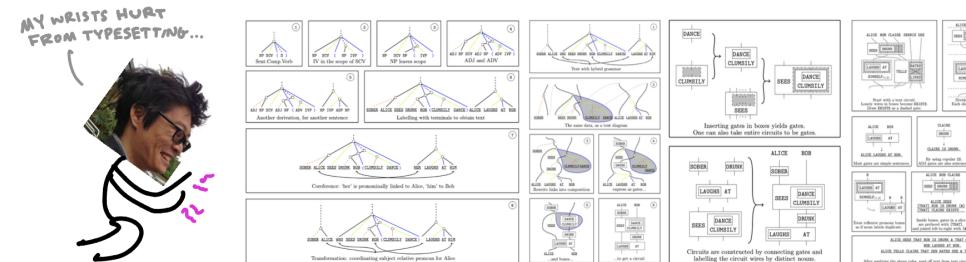
What are internal wirings?

Are text circuits a generative grammar?

What are text circuits good for?

These questions are what the rest of this thesis seeks to answer, one chapter at a time.

YOU SPOILT BRAT! BACK IN MY DAY BEFORE ALEKS MADE TIKZIT I HAD TO MAKE THOUSANDS OF DIAGRAMS OUT OF GARBAGE AND BLOOD! YOU THINK THIS IS HARD? FINDING A JOB IN QUANTUM FOUNDATIONS IN MY DAY WAS HARD! I WOULD HAVE KILLED FOR TIKZIT, AND THAT WOULD HAVE BEEN EASY BECAUSE I WAS A BOUNCER AND ONE NIGHT I BEAT UP THIRTY BASTARDS AT ONCE! HOW LAZY CAN YOU GET, AND YOU DON'T EVEN HAVE YOUR PHD YET? YOU WANT TO BE AN ARTIST? SUFFER! YOUR GENERATION DOESN'T APPRECIATE JUST HOW EASY YOU HAVE IT! BACK IN THE DAYS ...



$1\frac{1}{2}$

Corrections

There are several things I would like to get through in this long postscript, which has several purposes. In no particular order, the first is to settle most of the substantive corrections (recommended to me by my examiners Stefano Gogioso and Jules Hedges) in one go. The second is to situate this work with respect to formal linguistics as done by formal linguists. The third is to vent and find closure for myself.

$1\frac{1}{2}.1$ *Does it work?*

In the most important sense, no.

I will elaborate what I mean. There is a classical conception of structured approaches to AI/ML that permits capacities that go beyond what connectionist approaches are capable of, grounded in a kind of quasi-magical conception of mathematics as the ultimate form of understanding of — and hence control over — a phenomenon. It appears that part of the intellectual shock of generative AI is that one does not need to understand the mechanics of complicated phenomena in order to reliably induce them, given sufficient data and compute. This is obviously disappointing to many, and the highest sort of achievement that this thesis could have attained is to have been a source of hope. Even a year ago the prospects seemed grim, and part of what took me so long to write the introduction was to compromise the positive vision from some demonstration of the complete superiority of structured approaches (basically no longer tenable after the advent of ChatGPT with RLHF) to something weaker, such as a performant way to synthesise structural/symbolic and connectionist approaches that has some other kind of benefit, such as "interpretability" (whatever that means.)

So taking "*does it work?*" to mean "*does it justify the activity of mathematically formulating the structure of language with respect to non-intrinsic measures of value (such as practical application)?*", the answer is no, modulo most measures of value that people would care about. I say this because we've done some experiments.

Dear Reader,

How are you? I am well, thank you.

Forgive me for writing in this informal register; it is easier for me than the academic style (which I am no good at), and I would like to get these corrections done as painlessly as possible. You see, in the year or so it has taken for me to get around to these corrections, a lot has happened. I have gotten married, and (thankfully relatedly) I am soon to be a father. So I've learnt that there are more important things in life than a thesis, and I have otherwise been busy drawing diagrams about other things. While we are on apologies, please forgive me also for the overall tone of this work, which editorialises, and vacillates between serious and light; in my defense, I am constitutionally a trifler [SH05] (or else I wouldn't have gotten into formal linguistics), and I am still in the process of finding my voice as a writer.

I should preface by reminding you that all of this is written long after the rest of the work, and also that any opinions expressed here are strictly my own. I will also try my best to keep personal opinions tucked away in the margins so that they may be safely ignored.

$\frac{1}{2}$.1.1 Have experiments been done?

The approaches sketched out in Section 3.3 have been tried with neural networks in various ways by masters' students taking the Distributed Models of Meaning course offered in MFoCS here at Oxford, and separately it has been tried in the quantum setting by talented scientists and engineers at the company at which I am employed at the time of writing. In the classical case, it works for some bAbI tasks, but at the cost of hardcoding the structure of the queries, and it doesn't outperform a transformer. In the quantum case, it works for toy tasks, as it is generally difficult to get QML off the ground. Compositionality has not enabled any slam dunks so far.

I could be wrong; it could be that all of the experiments done so far were in compute and data regimes that are too small to be indicative of how these approaches may scale. It could be that rather than the level of words and sentences there is some structural benefit to be obtained at the textual level, and so on, but I am not personally holding out hope any more. I do maintain that the symmetric monoidal (and hence string-diagrammatic) approach to formalising the structure of language is the best and most natural way to synthesise the mathematics with modern (practical) machine learning, so the fact that it doesn't work all that well leaves me with a very dim view of everything else. As a result of these experiments and other experiences, my own view of the role of structure has been further demoted. As far as natural language is concerned, at worst such structure appears to be some crutch for us humans: inductive biases that constrain and unnecessarily burden more sophisticated algorithms that can deal with the complexity of "the real world". At best, it appears to be tradeoff: computationally cheaper but worse answers.

$\frac{1}{2}$.2 Can you situate this work with respect to the literature in formal linguistics?

Before I go on to do the scholarly work of relating my stuff to the stuff of others, I just want to say: look, it's not in the culture of my people (mathy folk) to read. It's either uninteresting so we put the book down, or if it is interesting, we put the book down and try to rederive it for ourselves. Accordingly, if armchair introspection was good enough for Harris and Chomsky and so on, I figured it was good enough for me too. It's hard for me to care about any formal linguistics because I'm just not sure that telling mathematical stories about language is a meaningful activity anymore. To be fair, I think there is a lot of interesting *Linguistics* out there — in my mind the kind of distinction between formal versus respectable linguistics is exemplified by the anecdote of Daniel Everett uncovering the structure of Pirahā in the field [Eve09], causing the formalist Chomsky to move the goalposts on what recursion means [FHC05] — but I have grown increasingly convinced that all of the effort to cast linguistics in mathematical terms up to now is best viewed as a kind of devotional or religious activity, a kind of benign way to kill time. Or perhaps it is a kind of crucible that refines the sensibilities of mathematical masochists. The only practical problem formal linguistics solves appears to be the gainful employment of overeducated fools such as myself.

One way to summarise things is that the technical contents of this thesis point towards a nearby counterfactual history where all the linguists in the "Garden of Eden period" [Par14] knew some of the modern structuralist mathematics that their programme obviously would have profited from. The fractiousness of linguists notwithstanding, it is my opinion that the interdisciplinarity of this thesis is accidental: from the perspective of the counterfactual history, the division of formal approaches to syntax and semantics (and some of the subdivisions thereof) would appear contrived. I am aware that this could come across as pretty arrogant and patronising talk, which is not my intention this time, because the point I am ultimately trying to make is that nothing in this thesis is particularly special, and could have been reinvented by anyone. Because I thought things through for myself for the most part, and since the relevant developments were due to collaborations with the similarly ignorant, text circuits and the other contents of this thesis owe no substantial intellectual debt to linguistics outside of perhaps Lambek and Firth, themselves far from the mainstream. Consequently, all of the parallels I am about to point out between the current stream of development and the main body of literature are not causally related. I take these parallels as indications of the "naturalness"

of ideas that could have come about anytime and independently of specific individuals, and accordingly as evidence for the "nearbiness" of the counterfactual history I am trying to gesture at.

$1\frac{1}{2}.2.1$ On Pregroups to Text Circuits vs. Transformational Grammar to Formal Semantics

FIRST COUNTERFACTUAL: TRUTH-CONDITIONAL VS. VECTORIAL SEMANTICS

Montague semantics may be essentially characterised as the meeting of two ideas [PH97]: structure-preserving maps from syntax, and taking truth-conditions to be the essential data of semantics. On some accounts, only the former aspect of compositionality of semantics according to syntax is essential [Sza00]. Accordingly, the first counterfactual is just the swapping of truth-conditional for vectorial semantics. Today there are several good reasons to prefer the latter over the former. First, the view that truth-conditions alone are the *sine qua non* of natural language meanings has been incompatible with correspondence theories of truth at least since Barr fixed Putnam's permutation argument [Put81, Bar], and it's not clear what other notion of truth would fit the bill. Second, vectors as lists-of-numbers are more expressive and computationally practical, so much so in its current form that the very need for a formal account of "semantics" is put to question. Third, (this one is just a personal opinion) with a rare few exceptions, the truth-conditional programme and its descendants are bankrupt, and worse, have terrible mathematical taste. A lot of mathematics has been marshalled to salvage the programme by trying to force intensions and pragmatics and everything-in-the-world into the propositional mould, and it is unclear what all of this mathematics buys us except for more of the same. In practical terms, the increasing extent to which statistical language models adequately handle semantics exactly matches the decreasing extent to which a complicated mathematical account of the same is warranted, and consequently, in theoretical terms it seems unserious to assert that the study of the mathematical models themselves lends insight into the phenomena they are intended to be surrogates for; if that kind of truth-seeking were truly the aim, wouldn't modern formal linguists be lining up to pick apart large language models?

But all this criticism can only be said with the benefit of hindsight, and to give credit, it all must have seemed like a very good idea at the time. A model-theoretic, truth-conditional account of semantic data was the natural choice for a concrete target for the structure-preserving map, I speculate, for several reasons: Montague himself was a logician, and truth-conditions were at once flexible enough to capture (to a logician's satisfaction) some semantic phenomena of interest, while being amenable to computation *by hand*, as was necessarily the case at the time owing to the lack of computers. Certainly there was adequate sophistication manipulating vectors by that time as well, but the vectorial view would have been more difficult to calculate with, even restricted to a setting without nonlinearities. It could be argued that, in any case, vectorial semantics in the form of word-embeddings requires a degree of data-storage and computing faculties that would not have been available until fairly recently. Moreover, even the theoretical soil was arguably unready:

This subsection about counterfactuals will also serve as an addendum to the cartoon literature review, which is folkloric, autochthonal, and maybe a little jingoistic: it is a caricature or myth of the field that we in it tell ourselves. I think that is sufficient for most readers, but if you are here, I owe you a critical and comparative retelling. In my view, the development of DisCoCat is only two minor counterfactuals removed from the lineage of mainstream formal semantics from Montague onto Heim & Kratzer and onwards. Moreover, both of these counterfactuals seem to rest only on the difference of when they began relative to the ambient development of mathematical formalisms and available computing.

category theory was insufficiently spread and understood, and hence a broadly accessible mathematical understanding of structure and structure-preservation outside of particular concrete instances was unavailable.

SECOND COUNTERFACTUAL: GENERATIVE VS. TYPELOGICAL VIEWS OF SYNTAX

In the origins of formal language theory taught in undergraduate computer science courses, a formal language is an acceptable subset of all possible strings from a stock of symbols. Context-free and context-sensitive grammars as rewrite systems are really generative in the everyday sense of the word, in that they are combinatoric abstract machines that produce acceptable strings. In the same way that pushdown automata parse context sensitive languages, typological grammars are another template for specifying grammars where parsing rather than production takes precedence; a proof-theoretic counterpart to combinatoric conceptions of grammar. Today "generative grammar" in some circles is just synonymous with "formal", which suggests a kind of unwarranted symmetry-breaking from path-dependency that could have also gone the other way if parsing formalisms such as typological grammars were more popular at the start. So in my personal mental model, there are (*inter alia*) two different kinds of mathematical formalisms for syntax, which are productive and parsing, and for any formal theory of syntax to gel properly with an analysis of communication, there's an onus on the modeller to provide a partnered formalism from the other side: for example, context free grammars as productive grammars get partnered with finite-state machines or, say, pregroup grammars that parse.

So the second counterfactual is just deciding to start from the parsing view rather than the productive one; to consider oneself a listener rather than a speaker. When it comes to natural language, perhaps the parsing and productive views would have been on more equal footing if computers were more advanced, because theory would have been held to account by the practical demands of serialising data structures as strings (production), and recovering them (parsing).

So if history had gone another way, quantum linguistics could have been the natural thing to do from the start, and it probably would not have required any detour through "quantum".

$1\frac{1}{2}.2.2$ On Montague's conception of grammar

In summary, to do "Montague semantics" means taking structure-respecting homomorphisms from grammar to meaning [JZ21]. Montague (likely) 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, which may be viewed as special cases of coloured PROPs. Hence text circuits share a mathematical lineage with many other mathematical conceptions of grammar, while also enjoying Montague semantics.

Montague semantics/grammar as Montague envisioned it is largely contained in two papers – *Universal Grammar* [Mon70], and *The Proper Treatment of Quantifiers in English* [Mon73] – both written shortly before

This subsection about Montague's grammar was originally a postscript titled *A modern mathematician's guide to Montague's "Universal Grammar"*. I've moved it here so that it can live alongside other things formal linguists might be interested in. I originally wrote this out of spite, because some category theorist said (flippantly, in my mind) that Montague semantics was "just a so-and-so". It is a kind of in-joke and shibboleth among category theorists to say that "X is just a Y" where X is quite pedestrian and Y is some scary arcane nonsense. Anyway, I thought this guy was full of it and didn't do his homework, so I thought I would go and show him up by getting the news at the source and rubbing his face in it. Though I still don't believe he did his homework, I did mine, and embarrassingly it turned out he was more or less right.

his mysterious death in 1971, so there were no opportunities for further elaboration. The methods employed were not mathematically novel – the lambda calculus had been around since the 1930s [Chu33], and Tarski and Carnap had been developing intensional higher-order logics since the 40s [Car88] – but for linguists who, by-and-large, only knew first order predicate logic, these methods were a tour-de-force that solved longstanding problems in formal semantics.

There is a natural division of Montague's approach into two structural components. According to Partee — herself 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 [PP08]:

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

More precisely, Montague devised a higher-order intensional logic for the first point, and the notion of a structure-preserving map from syntax to semantics for the second. The truth-conditional perspective was important at the time for enabling semantic computation, but within formal semantics there arose other perspectives on the nature of formal semantics, such as inquisitive [Inq] and update semantics [NBvV22]. Today, the empirical evidence we have from vector-based methods in computational linguistics is that none of those conceptions of semantics are intrinsically interesting or canonical: certainly none are procedurally necessary for a broad conception of practicality. So let's nevermind points 1 and 2b.

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

What remains that is of interest is the question of what Montague considered the structure of syntax to be. This is worth understanding, since we claim text circuits are a "structure of syntax", and that functorial interpretation of text circuits in symmetric monoidal categories is Montagovian semantics in spirit if not in letter. So let's begin. In Section 1 of *Universal Grammar*, Montague's first paragraph establishes common notions of relation and function – the latter he calls *operation*, to distinguish the n -ary case from the unary case which he calls *function*. This is all done with ordinals indexing lists of elements of an arbitrary but fixed set A , which leads later on to nested indices and redundancy by repeated mention of A . We will try to avoid these issues going forward by eliding some data where there is no confusion, following common modern practice. Next, Montague introduces his notion of *algebra* and *homomorphism*. I will shunt the reproductions of the definitions to the margin. First he separates the data of the carrier set and the generators from the

Definition 1₂.2.1 (Generating data of an Algebra). Let A be the carrier set, and F_γ be a set of functions $A^k \rightarrow A$ for some $k \in \mathbb{N}$, indexed by $\gamma \in \Gamma$. Denoted $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$

Definition 1₂.2.2 (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 1₂.2.3 (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 1₂.2.4 (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:

$$G((H_1(a), H_2(a), \dots, H_n(a)))$$

Definition 1₂.2.5 (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 1₂.2.6 (Homomorphism of Algebras). h is a homomorphism from $\langle A, F_\gamma \rangle_{\gamma \in \Gamma}$ into $\langle B, G_\gamma \rangle_{\gamma \in \Delta}$ when

1. $\Gamma = \Delta$ and for all γ , F_γ and G_γ agree in arity
2. $h : A \rightarrow B$
3. For all γ and lists of arguments $\langle \mathbf{A} \rangle$, $h(F_\gamma(\langle \mathbf{A} \rangle)) = G_\gamma(h(\langle \mathbf{A} \rangle))$

polynomial operations that generate the term algebra. All of Montague's algebras also come equipped with the data of *identities*, *constants*, and *composition*.

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

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. Here are several observations:

- (★) Because there is no condition of disjointness upon the X_δ — a view that permits the same word to play different syntactic roles — (1) permits the same basic expression $x \in A$ to participate in multiple types $X_\delta \subseteq A$.

Condition (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.
- (‡) some are typed multiply.

Taking a disambiguated language \mathfrak{U} on a carrier set A , Montague defines a *language* to be a pair $L := \langle \mathfrak{U}, R \rangle$, where R is a relation from a subset of the carrier A to a set PE_L , the set of *proper expressions* of the language L . It appears that a purpose of R defined in this way is 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".

It appears that Montague's 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 ME_L to be the image in 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 syntactic categories that "generate"). One consequence, in conjunction with (★), is that every multiply typed operation F induces a boolean algebra where the typings are the generators and the operations are elementwise in the inputs and output. Worse problems occur, as Montague's clone definition include all projectors, and when defined separately from the typing structure, these projectors may be typed in a way that permits operations that arbitrarily change types,

which appears to defeat the purpose. I doubt these artefacts are intentional, so I will excise my hermeneutical rights and assume his intent was a type-system with categorical semantics as we would use today. In so doing, we can summarise things fairly succinctly.

Proposition 1₂¹.2.7. Montague's grammars are coloured operads.

Proof. Definition 1₂¹.2.2 is equivalent to asking for all projections. Definitions 1₂¹.2.2 and 1₂¹.2.4 together characterise Montagovian algebras as (concrete) clones, which then generalised to (abstract) clones, which were then discovered to be in bijection with Lawvere theories [KPS14]; by an evident extension of [Prop 3.51] in that same paper 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 $(\Delta, \delta_0 : 1 \rightarrow \Delta)$. Lawvere theories are themselves a special case of operadic composition [Yau16]. Operadic composition naturally viewed as that of coloured trees, which is equivalently depicted as nesting expressions according to the tree-structure. Interpreting colours as types, operadic composition subsumes whatever one might wish to do with a typed gentzen-style sequent system where rules are multi-input single-output. \square

While typological grammars stop there, PROPs generalise operadic composition to multi-input multi-output, and as combinatorial specifications for string diagrams, weak n -categories generalise PROPs, and so we have shown weak n -categories to subsume a distinct evolutionary line of formal syntax from CFGs to TAGs.

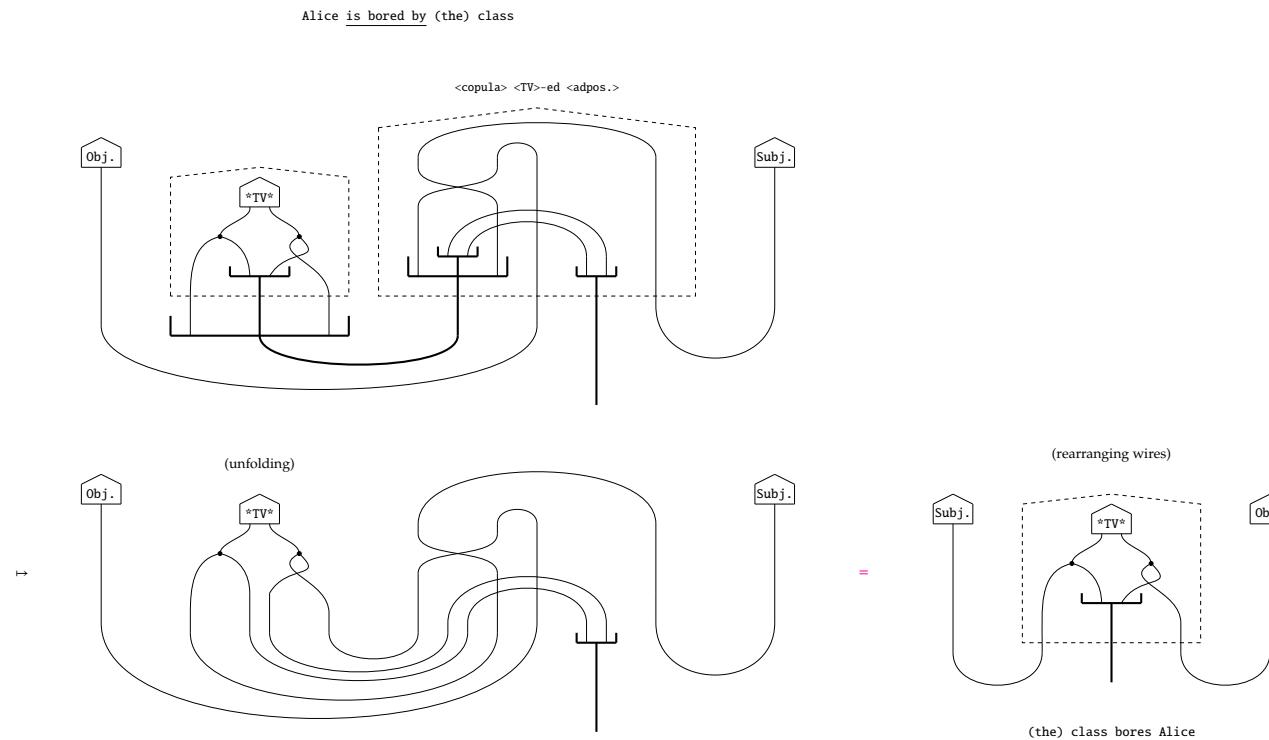
1₂¹.2.3 *On Deep Structure, the Universal Base Hypothesis, and the "Lexical Objection"*

As the story roughly goes [Har93], once upon a time some clever people dreamt up a *deep structure* that captured the syntactic relationships between the surface appearance of words in a sentence. Then they realised that by manipulating the deep structure with a *transformational grammar* (TG), they could relate sentences that seemed natural to relate. To be able to express the difference between related sentences as a single mathematical move was exciting stuff! Well, how far could one impose order on the jungle of syntax? Here's a thought: if language is perhaps something innate for humans (which as far as anyone knew was a reasonable assumption at the time), how about this for an ambitious idea: let's all try to find a single, canonical underlying mathematical structure that could be specialised to obtain a model for the syntax of any natural language: a *universal base* [PR69]. Of course, to do this, it was necessary to refine the mathematical models to more closely match the empirical data of English, so these clever people (who were now called formal linguists) got to work together, and that was exciting and fun. In those halcyon days, it was also believed that this could be done without a consideration of the meanings of words, because the assumption that grammar was somehow upstream of meanings, called *the autonomy of syntax* [Cro95], which is also a soft prerequisite assumption, if one holds that semantics is compositional according to syntax. But as TG got better at capturing English,

this belief about the autonomy of syntax started running into trouble because the line between meanings and grammar was blurry. So, some of the formal linguists decided to take semantics more seriously. But that meant that syntax alone wasn't going to cut it, and that meant throwing away some really nice ideas, like the autonomy of syntax, or (heaven forbid) the existence of a universal base that everyone was working hard to find. Some of the other formal linguists who really liked syntax didn't like the suggestion that syntax isn't special, and so they thought that taking semantics seriously was a silly idea and a bad move. And then everyone started saying mean things to each other, and then there was a big fight, and that was exciting and sad.

The fight is long settled, but I'm not sure that it's bedtime yet. My opinion is that the fight didn't really have to happen if they all just waited a while for computers to get better. I'm going to focus now on a particular example of how text circuits instantiated with vectorial semantics obtained from data might have patched things over, by sketching how one might address one of (in my opinion), the more deadly counterexamples to the autonomy of syntax, called the "lexical objection", which I'll explain now in grownup terms. One of the successes of TG was to demonstrate the essential sameness of sentences that look different on the surface. This can also be done by diagrammatic means: for example, a passive voice relationship like *Alice is bored by the class* and *The class bores Alice* is analysable as a topological equivalence of information flows

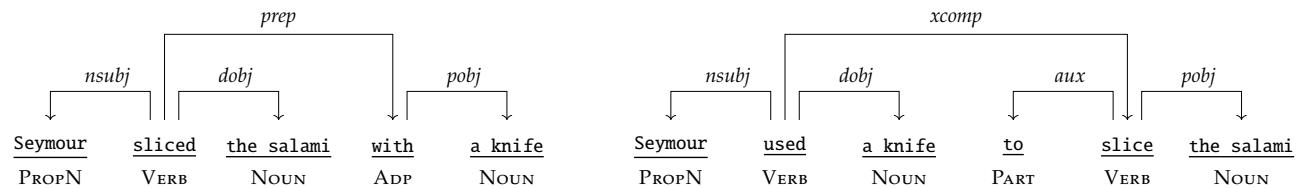
[CW21]



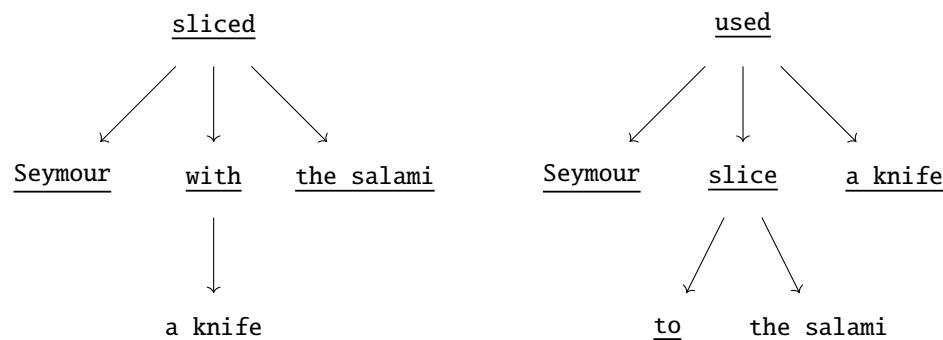
Whether it is TG or some other formal machinery that expresses it, if there is a universal base, then it probably ought to tell us about whether pairs of sentences of the kind above are really the same thing said differently, or else it wouldn't deserve such an important adjective like "universal". Now, if autonomy of syntax is true, then all of these grammar equations ought to be only concerned with the form or shape of the sentences involved, not on the particular words from your lexicon that are put in the place of subjects and verbs and objects and what-have-you. If there were such a lexical dependency, then one would have to posit special rules for grammar that are dependent on the meanings of particular words, so syntax wouldn't be autonomous.

The kicker is that there seem to be examples of such lexical dependencies, and that is the lexical objection.

Here is one example now, in the form of two sentences about Seymour making lunch [Lak68] along with their dependency grammar parses taken from [Dis]:



These are two sentences that seem to mean the same thing, and if there is a universal base and autonomy of syntax is true and semantics is compositional, then it had better be that there's a purely syntactic explanation for the equivalence, just like the cases of passive voices and relative pronouns and so on. Before we continue I want to head off a class of objections to this line of argument that says that there are differences in the meaning of these two ways of talking about Seymour, because by construing "meaning" to be suitably encompassing, there are for instance emotive or topical differences in the two sentences in the context of, say, a story or a poem. So let's call this the poetic objection. Taken to its logical conclusion, the poetic objection concludes that any two sentences that aren't exactly equal are meaningfully distinct, modulo a suitably generous interpretation of what meaning is. This trivialises semantics, insofar as semantics is about constructing equivalence classes on syntax, so we can be done asking questions right away. Just on the basis of what is more difficult and interesting and potentially informative, the current line of argument is preferable because seeks to distinguish differences in ways meanings can be "the same", even if this notion of sameness is grounded in essentially nebulous and intuitive judgements, because we might at least flesh out what the boundaries of those judgements are and learn something in the process. So let's agree to carry on, unfolding the syntactic structures of the sentences into something familiar and treelike:

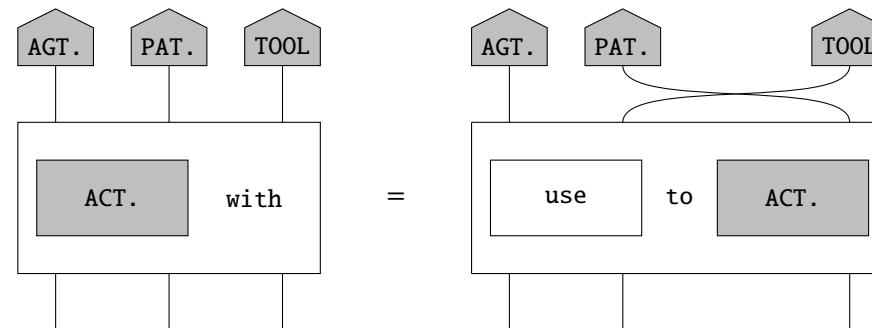


Now we have a problem: the root verbs are different, the shapes of the trees are completely different, and the leaf labels are juggled around. How would we transform one tree into the other? Whatever the method, it had better have something to do with the word *used*, because replacing *use* with some other word would change the meaning entirely. Fine, then let's try to save the autonomy of syntax by recategorising *use* to be

a grammatical function word, in the same league as copulas and pronouns and adpositions. Then we would be able to have rewrites relating the two trees above by fiat, but what would the general rule look like? It seems like the necessary and sufficient conditions to permit such an equivalence would have to boil down to something to something akin to a frame [FB01], that tells us when we have an agent performing an action on a patient with a tool, that the same as an agent using a tool to perform an action on a patient:

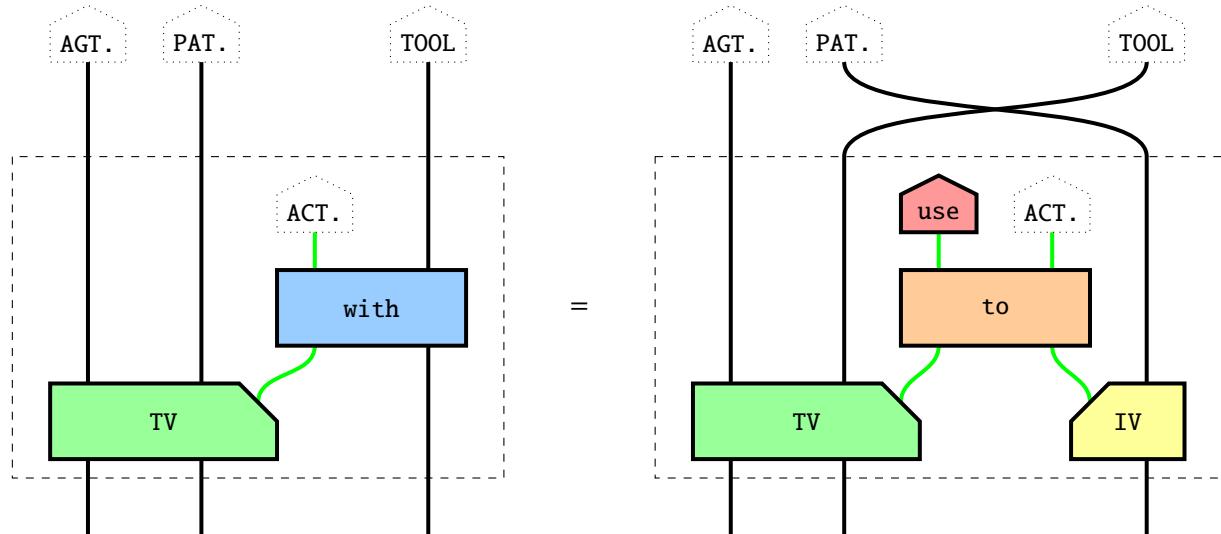
$$\left\{ \begin{array}{l} \text{AGENT ACTION PATIENT [with/using] TOOL} \\ \text{AGENT [uses] TOOL [to/for] ACTION PATIENT} \end{array} \right.$$

Wait a minute, isn't that fundamentally about meanings as we humans experience it in the world? How could the notion of tools be part of syntax? If we were to take this observation seriously, as in try to formalise these kinds of equivalences founded on such frames, then we would be retreading the path of generative semantics, cognitive linguistics, all the way to expert systems in GOFAI. But all of that would require potentially abandoning universal bases (who's to say that such frames are not culturally dependent) and the autonomy of syntax (if meanings can affect grammar, then that's akin to breaking the central dogma of biology: suddenly there's very little hope to untangle the phenomena because it's a mess of everything affecting everything). So you can probably appreciate why some die-hard formal grammarians didn't like this whole idea and why there was a big fight. Ok, now here is a sketch proposal to resolve the problem. Let's first turn everything both sentences into text circuits, like so, where boxes in grey indicate variable arguments:



Now let's interpret the nesting-boxes explicitly as parameterised operations, such that verb gates are factorised as verb-states on a verb type (green wire), and there is a separate wire (black) representing a noun type. The five coloured boxes represent five processes with representations we need to learn. More details of

this approach are given in Section 3.3, and separately elaborated in [RFL⁺24].



Now we can interpret the frame as an equality condition on these two circuits. If we instantiate all of the coloured boxes with neural networks, we can express the frame equality condition as a loss function to be minimised. In prose, we are asking that for all observed tuples of (agent, patient, tool) (as vector representations of nouns) and for all observed actions (as vector representations of verbs), the two composites of neural networks are equal. Concretely, let's say we have; a (assume metric) space of noun-representations N , where N_{role} indicates a subspace of representations corresponding to particular grammatical roles; a space for verb representations V with similar subspaces; an observed distribution from some text corpus $\mathcal{X} \sim N_{\text{AGT}} \times N_{\text{PAT}} \times N_{\text{TOOL}} \times V_{\text{ACT}}$; a divergence \mathbf{D} on $N \times N \times N$; four parameter spaces P corresponding to the parameters of the four neural nets TV , IV , with , to, and use ; and projection maps π_i to isolate outputs. The learning objective that corresponds to finding representations that satisfy the frame condition is then:

$$\inf_{p \in P_{\text{TV}} \times P_{\text{IV}} \times P_{\text{with}} \times P_{\text{use}} \times P_{\text{to}}} \left(\mathbb{E}_{(a,p,t,v) \sim \mathcal{X}} \left[\mathbf{D} \left(\left(\begin{array}{c} \pi_0(\text{TV}(a, p, \pi_0(\text{with}(v, t)))) \\ \pi_1(\text{TV}(a, p, \pi_0(\text{with}(v, t)))) \\ \pi_1(\text{with}(v, t)) \end{array} \right), \left(\begin{array}{c} \pi_0(\text{TV}(a, t, \pi_0(\text{to}(\text{use}, v)))) \\ \pi_1(\text{TV}(a, t, \pi_0(\text{to}(\text{use}, v)))) \\ \text{IV}(\pi_1(\text{to}(\text{use}, v)), p) \end{array} \right) \right) \right] \right)$$

We would still have to go find all the frames or somehow farm them from data, and if we wanted to not only be able to verify equalities of meaning but also be able to efficiently produce paraphrases we would have to solve the problem of circuit tomography in terms of known building blocks, and we would probably need a lot of compute to learn these representations well. But all these are technical rather than conceptual problems: this is a way to incorporate both kinds of paraphrase equalities, syntactic and semantic, in the same

mathematical framework. Taken in conjunction with vectorial semantics for lexicons which are a scalable solution for word-meanings, we have a compromise: here is something that works with modern technologies, and you import all of your formal grammar and formal semantics. There will remain autonomy of syntax to an extent, in that semantics is still compositional and syntax-directed but in such a way that learnt representations will also correspond to other syntactic representations in a meaning-sensitive way. Perhaps most importantly for those who care, this way of doing things lets us take both semantics and syntax seriously, and it leaves open the possibility that there does exist a universal base; by essentially outsourcing the additional complications of meaning to vectorial semantics informed by big data, we have — albeit in a different way — once again isolated the question of syntactic form on its own terms.

$1\frac{1}{2}.2.4$ On communication, and the mathematical infeasibility of the Autonomy of Syntax

Unless it is somehow unreasonable to analyse communication in terms of a producer and a parser, a form of the autonomy of syntax assumption is at odds with a form of compositionality of semantics: one of the two has to give. The thrust of Section 2.1 is that, even outside of the lexical objection, syntax must be constrained by a purely compositional semantics because of the constraints imposed by communication. I think this may be a genuinely new contribution to formal linguistics of a particularly nice sort, because it only really uses ideas that were already lying around. There are two branches of the development of Formal Linguistics writ large that I will attempt to relate this insight to here. The first is formal language theory and the relationship between formal languages and the abstract machines that generate them. The second is semantics in generative grammar.

The idea of tying together production- and parsing- machines for formal languages is well known, and is the conceptual basis of the Chomsky hierarchy: more complicated languages require more complicated parsing machines. See Figures $1\frac{1}{2}.1$ and $1\frac{1}{2}.2$.

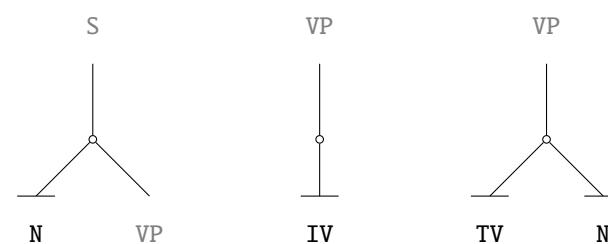
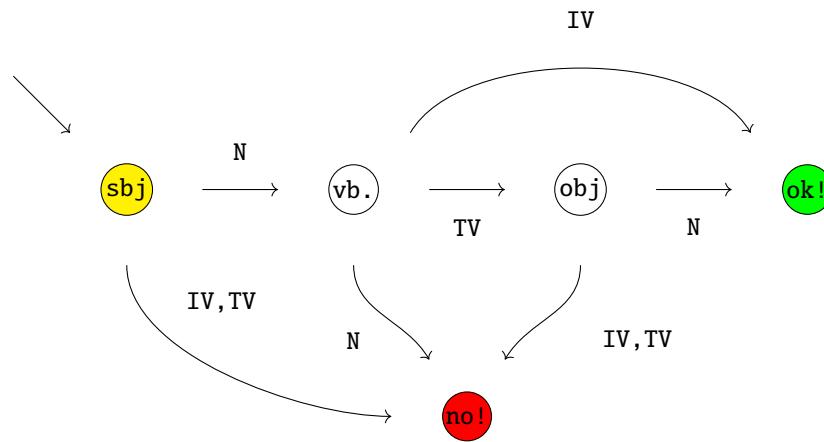


Figure $1\frac{1}{2}.1$: Here for example is a context free grammar for simple sentences that involve either a single transitive or intransitive verb. I've gathered together terminals, depicted as feet, into the basic generators.

Figure 1 $\frac{1}{2}$.2: And here is its corresponding finite state machine. States are depicted as nodes, and transitions are depicted as labelled directed edges. This machine operates over the alphabet {N, IV, TV}, and it only accepts those strings that are producible by its paired CFG.



But there is something missing in this picture, and that is semantics. The finite state machine in this example only does a weak form of parsing, indicating whether a string is acceptable or not. One obvious way to incorporate semantics on the parsing side here is to upgrade the finite state machine into a variant pushdown automaton that simultaneously handles the transcription of the truth-conditional semantics in the λ -calculus. See Figure 1 $\frac{1}{2}$.3.

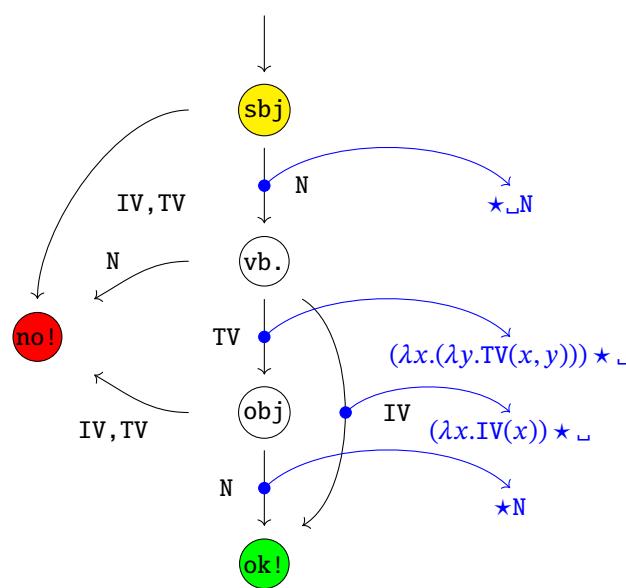


Figure 1 $\frac{1}{2}$.3: The blue arrows branching off of the transitions indicate what rewrites to perform on a side-buffer, where \star denotes whatever was previously in the buffer, and \sqcup indicates the placement of a cursor position where future rewrites are to take place. We're overloading notation here so that terminal symbols N , IV , TV in the buffer correspond to typed elements and functions $N \in D_e$, $IV \in D_{\langle e,t \rangle}$ and $TV \in D_{\langle e,\langle e,t \rangle \rangle}$ respectively, with respect to some bestiary of individuals/entities D_e and truth values D_t and their type closures in **Set** under exponentiation.

Example 1 $\frac{1}{2}$.2.8. An example stack-trace of the operation of the machine of Figure 1 $\frac{1}{2}$.3 on the input Ann loves Jan is:

$$\varepsilon \xrightarrow{\text{Ann}} \sqcup \text{Ann} \xrightarrow{\text{loves}} (\lambda x.(\lambda y.\text{loves}(x,y))) \text{ Ann} \xrightarrow{\text{Jan}} (\lambda x.(\lambda y.\text{loves}(x,y))) \text{ Ann Jan}$$

Afterwards and separately, the application rule of the λ -calculus yields the desired expression to be evaluated with respect to a model, namely: $\text{loves}(\text{Ann}, \text{Jan})$.

Now wait a minute, we also want semantics to be compositional, guided by the structure of syntax. But all of that structure is in the CFG. It's well understood how to do semantics for generative grammar on that end [HK98], but now we're left with a technical problem: how do we know whether the pushdown construction gives us the same semantics as the one we would usually want for the CFG? If we had that, then we would have rolled together a parser and a producer sharing the same semantics, which would be a particular mathematical model for communication. We could perform a case analysis on this example to achieve this, but that approach won't scale, and it seems pretty thorny to generalise because the whole pushdown idea operates on λ -expressions in linear form, which forgets all of the nice syntactic structure given by the CFG. If we wanted to do this systematically while respecting compositionality of semantics, it seems that the conceptually cleanest way to go about it is to look for a kind of strong equivalence: we can try to make it so that every generator

on the productive side corresponds precisely to an appropriate bit of partially composed semantics on the parsing side. But in this example, what are we to do with the ansatz types S and VP that live in the CFG but don't occur on the parsing side? The fundamental conceptual obstacle here is that deep structure (should it exist) potentially has access to all kinds of extra labels and structural information that the parsing side doesn't have access to; the latter can only see a string of terminals. Let's take a step back here and start over.

Maybe starting with a finite state automaton on the parsing side is not the best beginning, because it only cares about whether a string is acceptable or not. If instead we used a typological grammar that witnesses judgements of grammaticality along with proofs as witnesses, we could exploit the fact that the proofs themselves reflect grammatical structure. Now Section 2.1 is well-motivated with respect to the literature, so you can go see how this line of investigation plays out over there.

$1\frac{1}{2}.2.5$ On frameworks for rewriting systems

I think that Section 3.2 is pretty much well-motivated with respect to the literature already, so there's just one conceptual issue I want to address here. One of my examiners noted that it seems like bad mathematical taste to use free infinity-categories for what basically amounts to graph rewrites, which raises the obvious question: why not just use graph rewrites instead? First, I agree with his comment about taste, but I felt in this case it could not be avoided, because second, the answer lies in Figure 3.24, and it has to do with locality and long-range dependencies. I'll try to elaborate the issue in prose here. Tree-structures represent context-free symbol-rewrite systems, and transformational grammar and tree-adjoining grammars are about rewriting trees. Of course, symbol-rewrite systems can be context-sensitive, and in the same vein, we might expect that tree- or graph-rewrite systems can also be context-sensitive. Now here is where things stop being pedestrian: suppose we have a long-range dependency of some kind between A and B, where it doesn't really matter what happens in between. An example is coreference in text. This is a kind of *context-insensitivity*, where rather than a rewrite requiring a context of a particular form, the surrounding and intervening context may freely vary to a degree. Designing an abstract machine that obeys these constraints is tricky because rewrites could in principle require evaluating whether some predicate holds for the contents of memory before executing a rewrite, as opposed to a simple lookup. Concretely, in the case of strings, context-insensitivity already requires more than just a stack-memory; it would require a Turing machine that could also operate on that memory to determine whether the context is of the right form. There's a whole host of issues here: technically it's not clear what such a machine would look like if we go from strings to graphs, and conceptually if we had such a machine we would have to craft custom limitations on the kinds of well-formedness predicates that permit rewrites so that we could stay in an efficiently parseable class of languages, on pain of our machinery overshooting the human capacity for natural language. The solution to these problems that I have opted for here is to keep all rewrites local and properly context-free for graphs, but I let the machinery of infinity categories handle bookkeeping for handling spatial rearrangements of the deep structure so that it is

possible to redefine what locality means. The price to pay here is that one must explicitly keep track of not just the connectivity of the graph at any given time but also its spatial arrangement, but this is also potentially a feature: the difficulty of finding the right arrangement to go to next conceptually maps onto the computational difficulty of understanding natural language (but I won't commit to any sort of mathematical realism about it). I'll also remark that the machinery of Section 3.2 can be reversed to go from circuits to text, which can be piped together with the machinery of Section 2.1 to go from text to circuits to also give a satisfactory model of communication, though what I have not done in this case is demonstrate strong equivalence, which I will leave as an open question for now.

$1\frac{1}{2}.2.6$ On formality in cognitive semantics

There are two conceptual contributions here to formal linguistics. First is the demonstration that, in principle, we may interpret the meanings of words to be instructions about how to set up and move shapes around in cartoon sketches of conceptual schema, without sacrificing rigour. I do this by defining the symmetric monoidal category **ContRel** of topologically continuous relations, which serves as a mathematically formal and compositional setting for cartoons. I've gone to some length to convince the reader that, modulo a little elbow-grease, anything one can linguistically describe happening to assemblages of shapes can be modelled in **ContRel**, so we may conservatively generalise Montague semantics — construed as a symmetric monoidal functor from text circuits as an underlying deep structure to some other symmetric monoidal category as semantics — from truth-conditions to shapes.

Second is a computable (by hand) mathematical model of metaphor. The core idea is simple: if we consider a metaphor to be a pair of structure-preserving maps from the data of a conceptual schema C to two different semantic settings X and Y , so long as the maps satisfy certain algebraic properties, we may start from an expression in X , go backwards along one of the maps to C , and then travel forwards along the other map to Y to obtain another expression. The nature of the algebraic properties makes it so that travelling backwards along a map is not only well-defined, but the resolution of where we end up in the preimage also doubles as bookkeeping for different choices about how to interpret the metaphor. By starting with X as text circuits as a model of deep structure, encoding the data of a conceptual schema C as freely generated by the signature of a text circuit with equational relations, and taking Y to be the setting **ContRel**, we may compute the meaning of *Vincent spends his morning writing*, via the metaphor *TIME is MONEY*, as a vignette of pictures that represents the conceptual situation in much the same way that moving meeples around different zones in a boardgame encodes the state of the game. This sort of boardgame-semantics is then a model that serves as a further basis upon which one can do truth-conditional semantics. What is quite cute about all of this is that it is the interpretation of metaphors as pictures, using picture-maths.

These two contributions address — to my knowledge — several conceptual lacunae in both truth-conditional and vectorial semantics. The first such gap is that we use language in the first place to construct the underly-

I will confess that what truly motivated all of the effort in this thesis was the attempt to put metaphors on formal footing, in pictures. There is an attitude that the real mensch logic and graphs that formal linguists are fond of is somehow better by virtue of mathematical rigour than the fluffy stuff cognitive semanticists are interested in, and that didn't sit right with me. I think I've achieved what I set out to do, but in such a way that my zeal for formal linguistics annihilated itself: I was moving little blobs and stickmen around with words in a compositional fashion to create cartoon vignettes, which was really no different than programmatic animation with pen and paper under doctrinal constraints that no longer in my mind usefully restricted the nature of the kind of inquiry into language that was intended. What started as mathematical modelling in search of something essential became in myself a dawning awe of the vastness of language, and the recognition that I had both the logician's and the algebraist's sicknesses in my spirit. How I would attempt to describe them is that the logician's sickness is the desire to subjugate and quarantine the terror of reality within logos, and the algebraist's sickness is the desire to construct abstract machines that will solve the problem in a way that absolves one of having to make choices. These are, I believe, the fear of life and of living, respectively. If I went back in time, I don't think I would have been able to talk myself out of it. From discussions with other postrationalists, it seems that the sickness doubles as the sole cure: one just has to really take rationalism seriously and come out the other side with the sensibility of an artist, burdened and blessed with the responsibility to choose what is meaningful for themselves. Before, from pride, I looked upon mathematical linguistics with disdain; now, from humility, pity.

ing models with respect to which we might eventually apply the truth-conditional view, for instance when we tell a story, or when we present a reasoning problem in mathematics. This appears to require a discourse-theoretic approach where composition of the underlying models that truth-conditions are evaluated against, and the machinery in this thesis appears to suffice in principle. I don't doubt that there are frameworks that meld together truth-conditions discourse-theoretically; the conceptual distinction of this approach is that the composition of the underlying models (as states), meanings of words in text (as updates), and of truth-conditions (as tests) all sit on equal footing, and all moreover in a fashion that generalises beyond truth-conditions to admit broader conceptions of what the essential data of semantics might be, as the modeller wishes.

A second gap, particularly a challenge for the vectorial view but more generally pressing, is a compositional semantics for topological linguistic relationships, such as containment, which motivated and justified introducing the machinery of topologically continuous relations. Evaluating/interpreting the meanings of such topological words required concrete models to evaluate against, which necessitated a construction of a compositional setting in which one could hold onto and label "non-pathological" shapes, and a bit of mathematical effort was expended to show that such collections of shapes corresponded to conservative generalisations of special frobenius algebras, which end up behaving graphically simply as wires that permit splitting and merging. That effort in my view distinguishes this approach from others in that it also sought to demonstrate an ideological point: that pictures are all you need.

The third, which is a gap for any compositional account of natural language semantics guided by syntax, is that one needs something else around in order to make sense of metaphors. For this, I argue that spans of functors — of essentially the same kind used in my analysis of communication — appear to be a promising foothold, and I do believe I am the only person thus far to have computed a pictorial semantics of a metaphor by formal means "all the way", starting from the basic syntactic structure. There are many things I am unhappy about with this thesis, but I am at least happy about that.

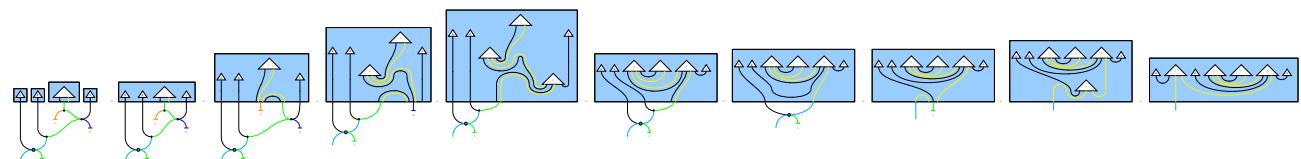
2

On communication

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

2.1 How do we communicate using language?

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



HERE ARE SOME NAÏVE OBSERVATIONS ON THE NATURE OF SPEAKING AND LISTENING. Let's suppose that a speaker, Charlie, wants to communicate a thought to Dennis. Charlie and Dennis cooperate to achieve the miracle; Charlie encodes his thoughts – a structure that isn't a one-dimensional string of symbols – into a one-dimensional string of symbols. And then Dennis does the reverse, turning a one-dimensional string of symbols into a thought-structure like that of Charlie's. It may still be that Charlie and Dennis have radically different internal conceptions of what FLOWERS or GIVING or BEETLES IN BOXES are, but that is alright: we only care that the *relational structure* of the thought-representations in each person's head are the same, not their specific representations.

THE NATURE OF THEIR CHALLENGE CAN BE SUMMARISED AS AN ASYMMETRY OF INFORMATION. The speaker knows the structure of a thought and has to supply information or computation in the form of choices to turn that thought into text. The listener knows only the text, and must supply information or computation to deduce the thought behind it. By this perspective, language is a shared and cooperative strategy to solve this (de/en)coding interaction.

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

Alice likes flowers that Bob gives Claire.

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

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

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

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

We assume Charlie and Dennis speak the same language, so both know how words in their language correspond to putative building blocks of thoughts, and how the order of words in sentences and special grammatical words affect the (de-/re-)construction procedures. Now we have to explain how it is that the two can do this for infinitely many thoughts, and new thoughts never encountered before. Using string diagrams,

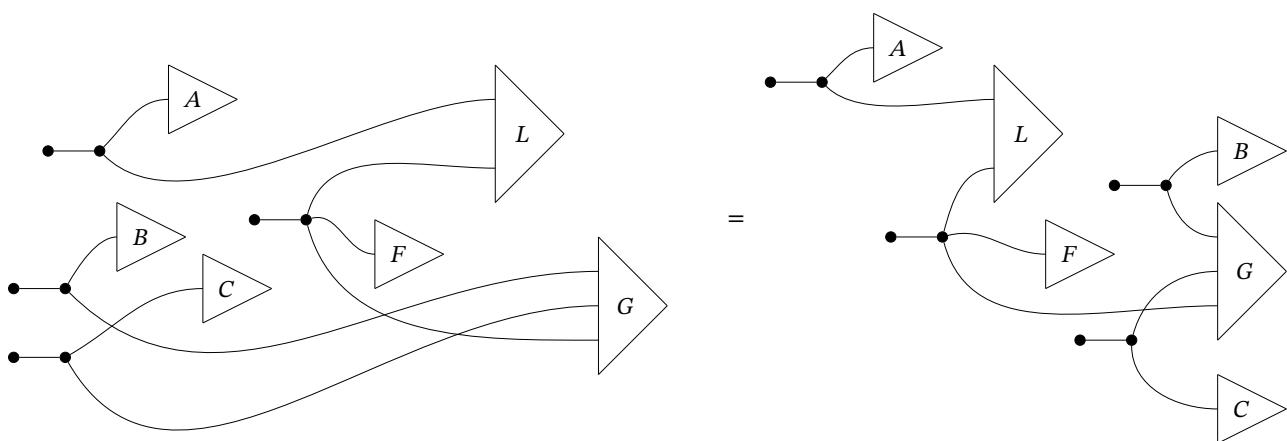


Figure 2.1: Charlie and Dennis agree on the conceptual organisation entities and relations up to the words for those entities and relations. Just as a running example that does not affect the point, let's say we can gloss a thought in first order logic as $\exists a \exists b \exists c \exists f : A(a) \wedge B(b) \wedge C(c) \wedge F(f) \wedge L(a, f) \wedge G(b, c, f)$. In diagrammatic first order logic [HS20], this is equivalently presented as the following diagrams (and any other diagram that agrees up to connectivity.) For example, Charlie could ask Dennis comprehension questions such as WHO GAVE WHAT? TO WHOM?, and if Dennis can always correctly answer – e.g. BOB GAVE FLOWERS. TO CLAIRE. – then both Charlie and Dennis agree on the relational structure of the communicated thought to the extent permitted by language.

this is surprisingly easy, because string diagrams are algebraic expressions that are invariant under certain topological manipulations that make it easy to convert between different shapes of language.

Example 2.1.1 (Alice likes flowers that Bob gives Claire.). Let's say Charlie is using a context-free grammar to produce sentences, and Dennis a pregroup grammar.

Figure 2.2: The rule of the game is that Charlie and Dennis can agree on a string-diagrammatic encoding strategy before having to communicate with each other. Here is one such strategy. Charlie might generate the example sentence as depicted.

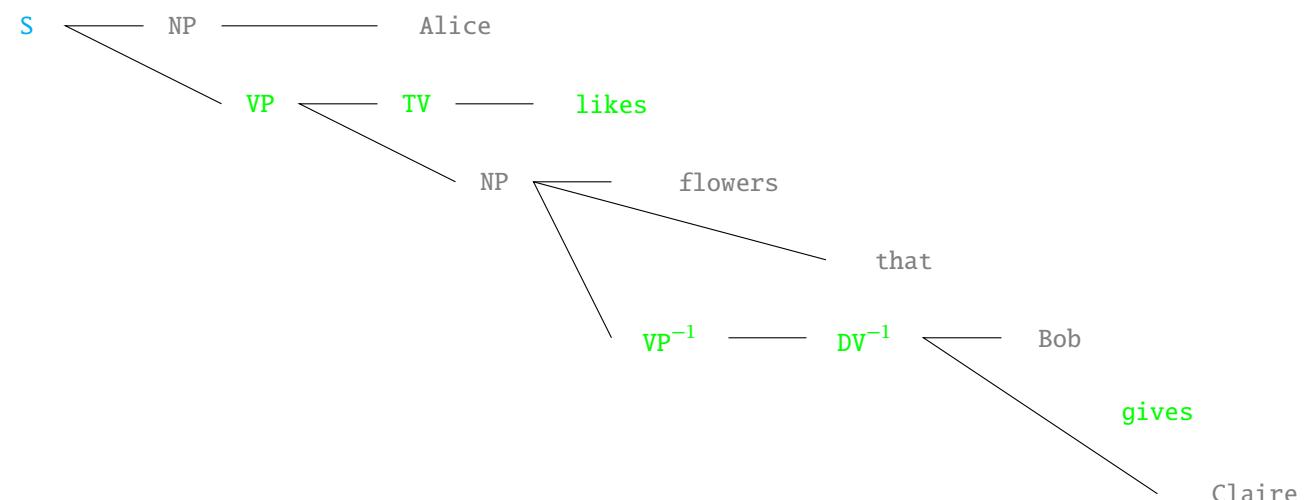


Figure 2.3: Mathematically, it makes no difference if we take the Poincaré dual of the tree, so that zero-dimensional nodes become one-dimensional wires, and branchings become zero-dimensional points linking wires – but we can just as well depict those points as boxes to label them more clearly.

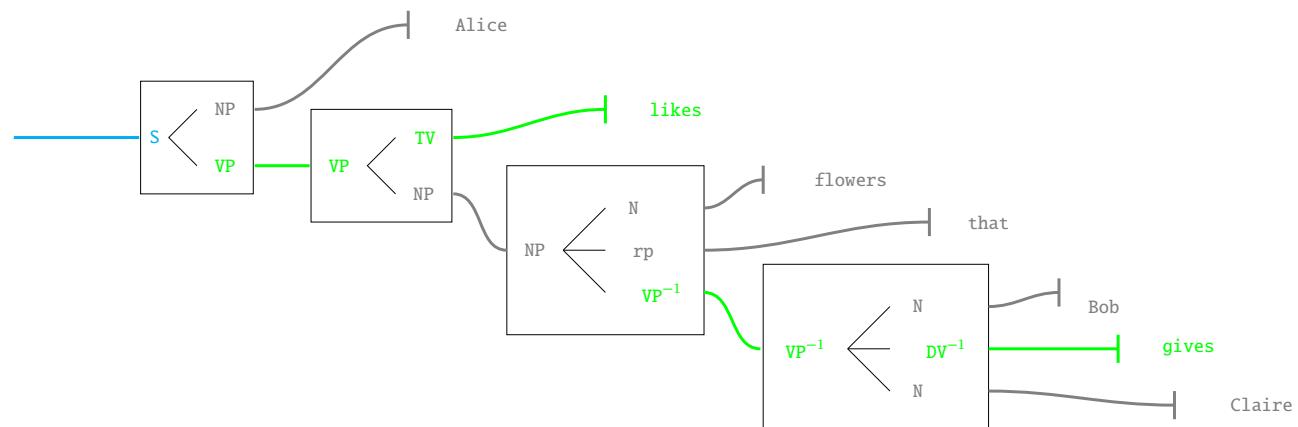
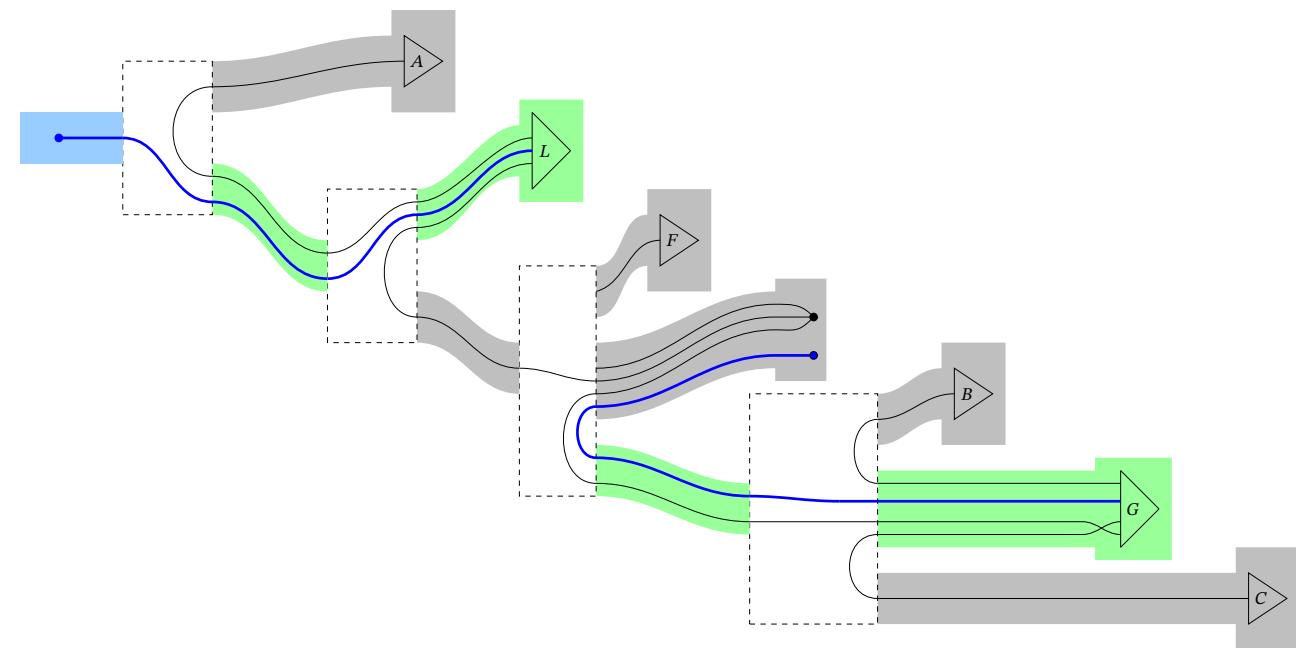


Figure 2.4: Now that Charlie can express their grammatical structure string-diagrammatically, they can try to deform their first-order-logic diagram – representing what they mean to communicate – subject to the constraint that every one of their branchings (the structure of the CFG) is something recoverable by Dennis using just pregroup reductions. To do so, Charlie introduces a formal blue wire to mimic Dennis's sentence-type, and stuffs some complexity inside the labels in the form of internal wirings: a multiwire configuration for that, and a twist for gives. Those internal wirings are the content of Charlie and Dennis's shared strategy. In passing, I'll remark that by the outside-in convention for functor boxes 2.10, this diagram constitutes a monoidal functor from this particular CFG to pregroup diagrams, where nonlabel tree-nodes are partial monoidal closure evaluators. Replacing rigid autonomous closure with cartesian closure and n, s with e, t recovers montague semantics for CFGs (c.f. Curry-Howard-Lambek correspondence for the case of typed lambda-calculus and cartesian closed categories, and all of Heim and Kratzer [HK98]), and interpreting the closure in a compact closed setting recovers montague semantics for CCGs [YK21].



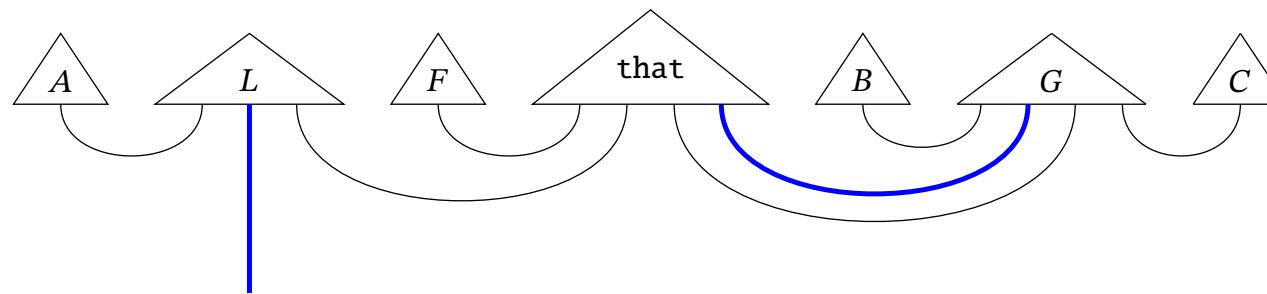


Figure 2.5: So, when Dennis receives the sentence, Dennis's pregroup derivation yields a pregroup diagram that is connectively equivalent to what Charlie stuffed inside the context-free grammar structure. So now the two have strong equivalence between their grammars in the sense that every one of Charlie's branches is resolved by one of Dennis's reductions. As is convention for pregroup diagrams, we only use types n and s – the latter denoted by a blue wire here – and we'll leave the directionality (rigid autonomous turning number) of wires implicit, so you can either trust me that everything typechecks or do it yourself.

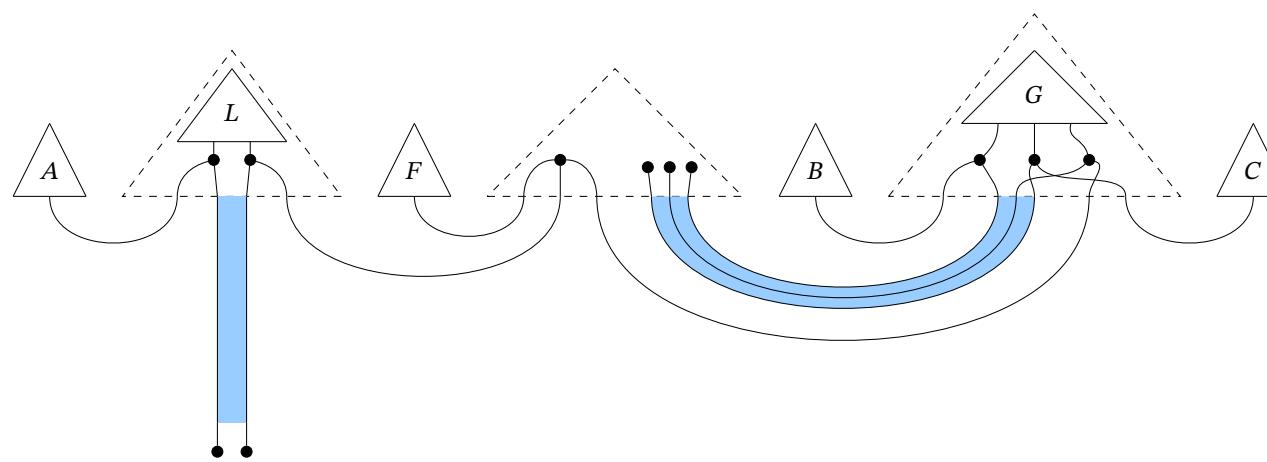


Figure 2.6: Now to fully recover Charlie's intended FOL-diagram, Dennis refers to the internal wirings from their shared strategy, and fills those in.

Example 2.1.2 (Alice likes flowers. Bob gives Claire (those) flowers.). Now we try the same content as the previous example but presented as a text with two sentences.

Figure 2.7: Charlie's diagram morphed to fit a text circuit. The dotted blue line is a formal mark to indicate a sentential boundary. Observe how new discourse elements are introduced as states, and how open wires correspond to ongoing discourse and deletions mark completed discourse. This diagram also indicates that text circuits can be given semantics in FOL.

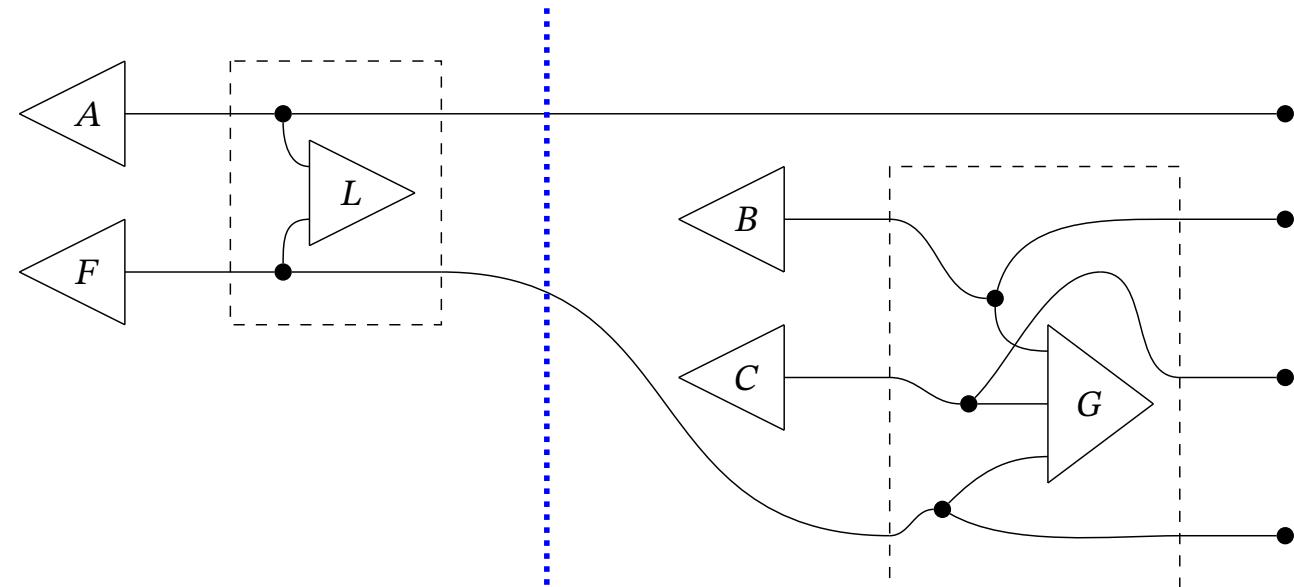


Figure 2.8: Dennis already knows how to parse individual sentences to extract the FOL using internal wirings. Observe there is a mathematical complication that arises in determining how many noun-wires should go into the sentence wire-bundle; we need to account for this later.

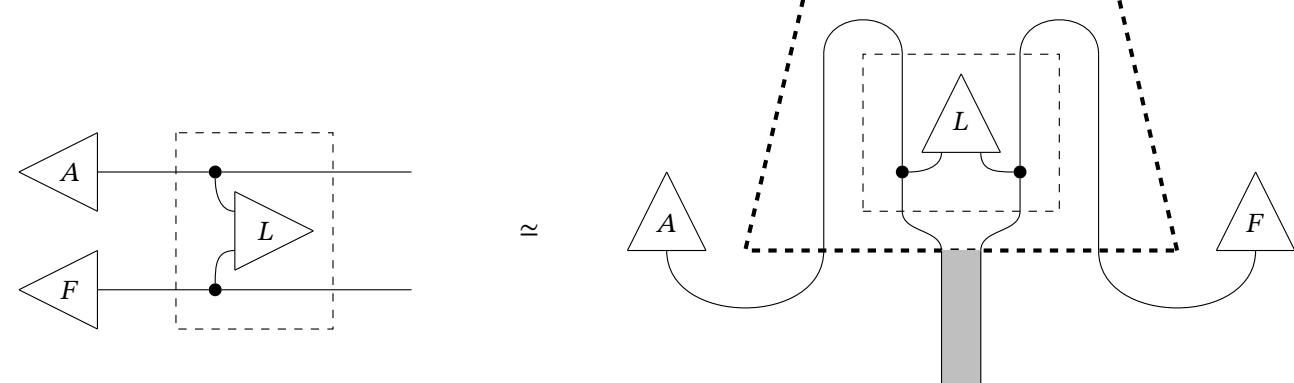
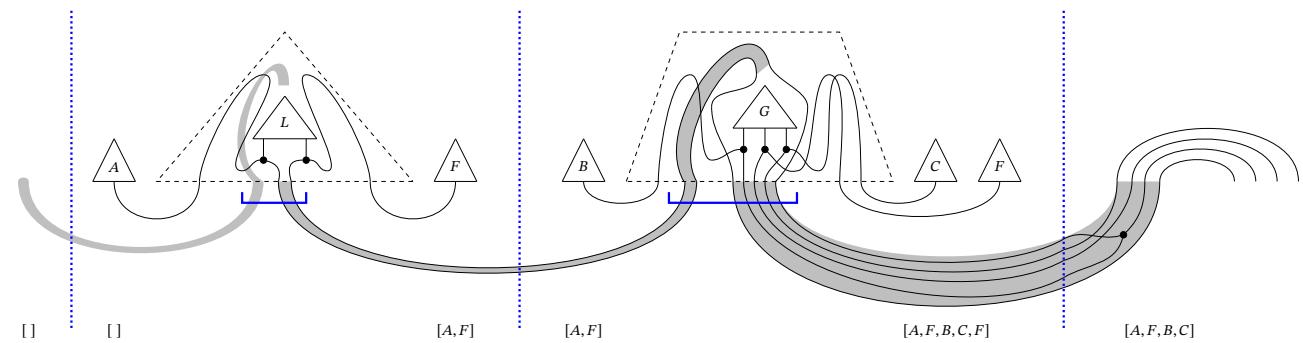


Figure 2.9: To deal with text, Dennis can pass a growing bundle of sentence wires along horizontally.

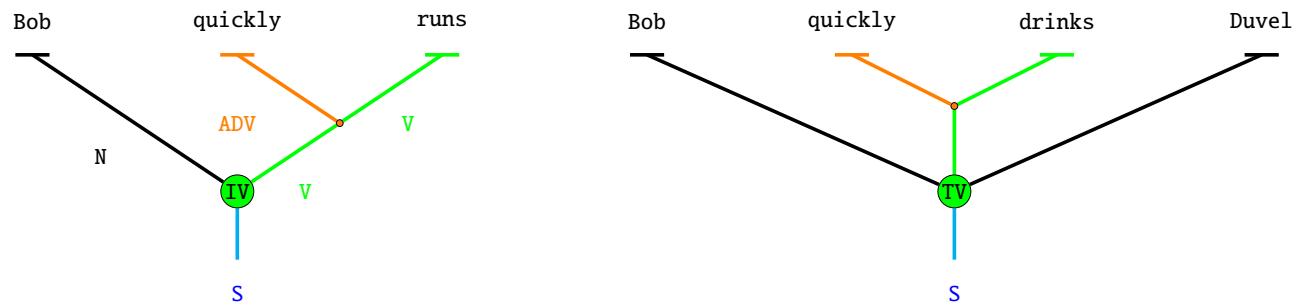


THE ESSENCE OF INTERNAL WIRINGS: The relational content that is communicated by language is not inherently one-dimensional, but must be encoded in and decoded from the one-dimensional strings of language. Internal wirings [WC] provide a way to approach this coding problem topologically: while productive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

2.1.1 An issue with functorial semantics of internal wirings

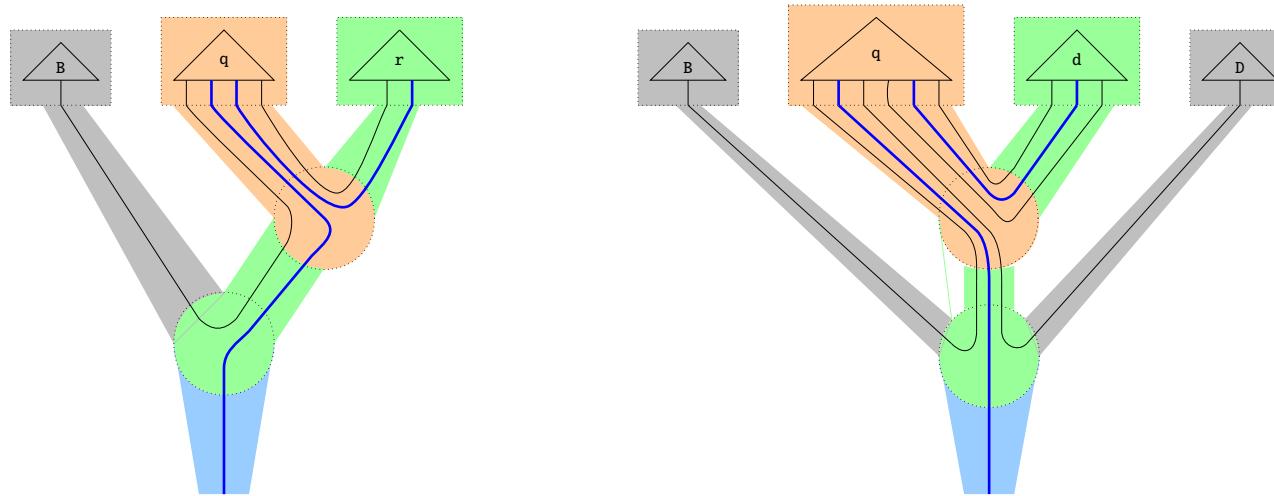
THERE IS A MATHEMATICAL ISSUE: the "filling in" of internal wirings is not *in general* functorial, for either speaker or listener. The exemplified issue is that sometimes the particular internal wiring depends on what words are around it.

Example 2.1.3 (Nonfunctoriality of internal wirings for productive grammars). Let's consider an easy context-free grammar as in Figure 2.1.3, with just four types and three rules apart from labels. The types are: S for sentences, N for nouns, ADV for adverbs, and V for verbs. There is a single adverb introduction rule, and two verb introduction rules for intransitive and transitive verbs.

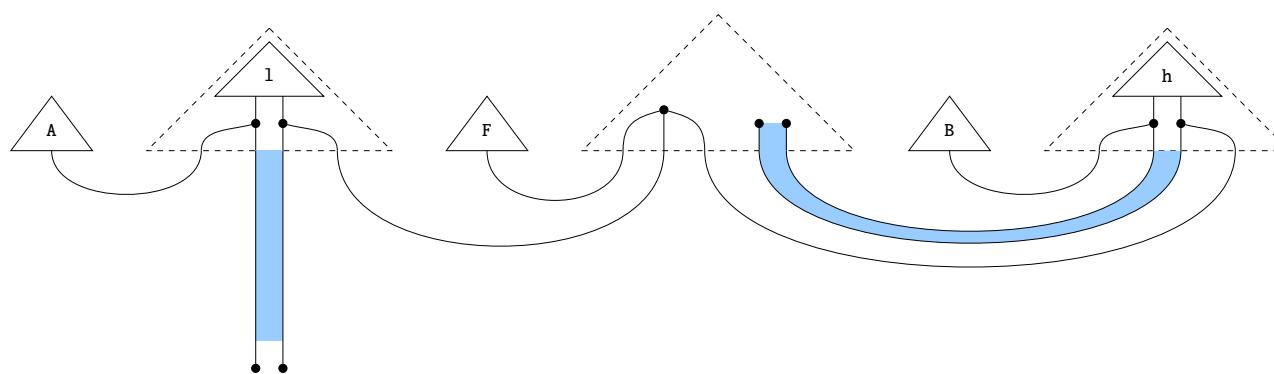


Now suppose we want to describe a functor from this context free grammar to a pregroup grammar with just types n and s . We know how verb states ought to look, and we know that adverbs ought to modify a verb. We can get pretty close with a first sketch, depicting the desired action of the functor using the outside-in convention for functor boxes, and we can slim them down to tubes. Now the simplicity of the CFG reveals a complication. Since there are two possible kinds of verbs, there are two possible kinds of adverbs, and accordingly two possible kinds of adverb introduction rules. A functor from the CFG to a pregroup diagram

can't send the single adverb introduction rule to two different things at the same time.



Example 2.1.4 (Nonfunctoriality of internal wirings for parsing grammars). Compare Alice likes flowers that Bob hates to the sentence in Figure 2.6; here the object relative pronoun that is connected to a transitive verb hates rather than a ditransitive gives. The internal wirings work fine in this example, but now that deletes two wires instead of three; a functor can't map the same word-state to two possible instantiations.



2.2 Discrete Monoidal Opfibrations

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

Scholium 2.2.1. Expressing the coherence conditions of monoidal functors using equations involving functor boxes as below is not new [Mel06]. The idea of a functor being simultaneously monoidal and a opfibration is also not new. What is new is minor: the express requirement that the lifts of the opfibration satisfy interchange, which is in general not implied when a functor is both monoidal and a discrete opfibration.

Figure 2.10: There are two conventions for depicting the action of a monoidal functor on parts of a string diagram. The first follows source-to-target *outside-in*. This convention is used for other work in internal wirings, since it is well-suited for describing functors that send atomic generators in their domain to more complex diagrams in their codomain.

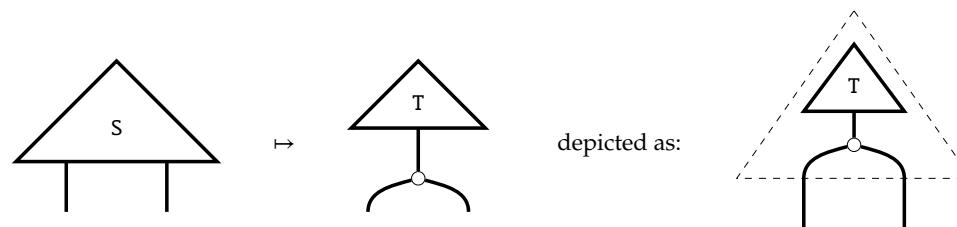


Figure 2.11: The other convention is *inside-out*. For the following section, we will define the coherence conditions of discrete monoidal opfibrations using this convention.

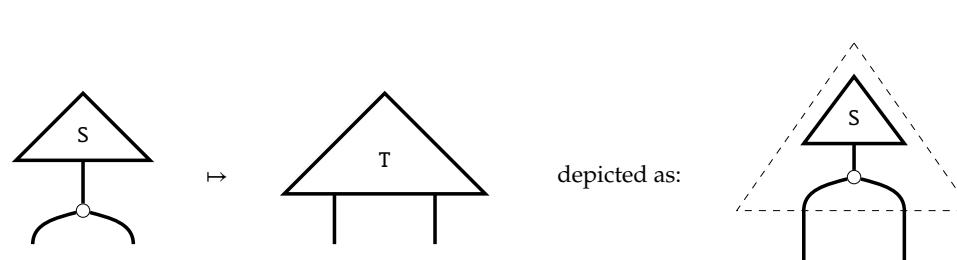


Figure 2.12: Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$. Then we have this diagrammatic representation of a morphism $\mathbf{F}f : \mathbf{F}A \rightarrow \mathbf{F}B$ in \mathcal{D} .

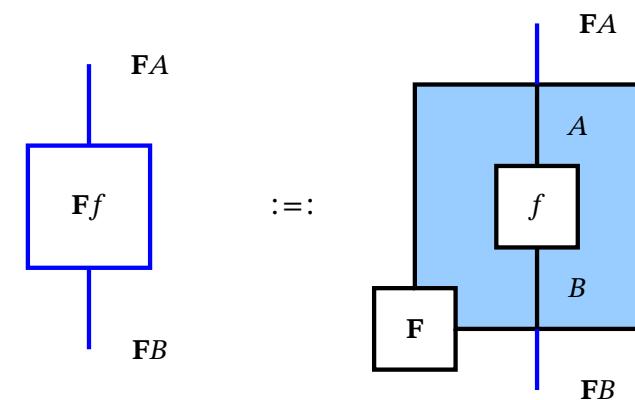


Figure 2.13: The use of a functor box is like a window from the target category \mathcal{D} into the source category \mathcal{C} ; when we know that a morphism in \mathcal{D} is the image under \mathbf{F} of some morphism in \mathcal{C} , the functor box notation is just a way of presenting all of that data at once. Since \mathbf{F} is a functor, we must have that $\mathbf{F}f; \mathbf{F}g = \mathbf{F}(f; g)$. Diagrammatically this equation is represented by freely splitting and merging functor boxes vertically. **N.B.** sequential merging of two boxes requires that the two wires to-be-connected within the boxes – in this case labelled B – need to be the same; a case where merging is disallowed is when $\mathbf{F}f; \mathbf{F}g$ typechecks in the outside/target category, but $f; g$ does not in the inside/source category because the functor identifies nonequal wires.

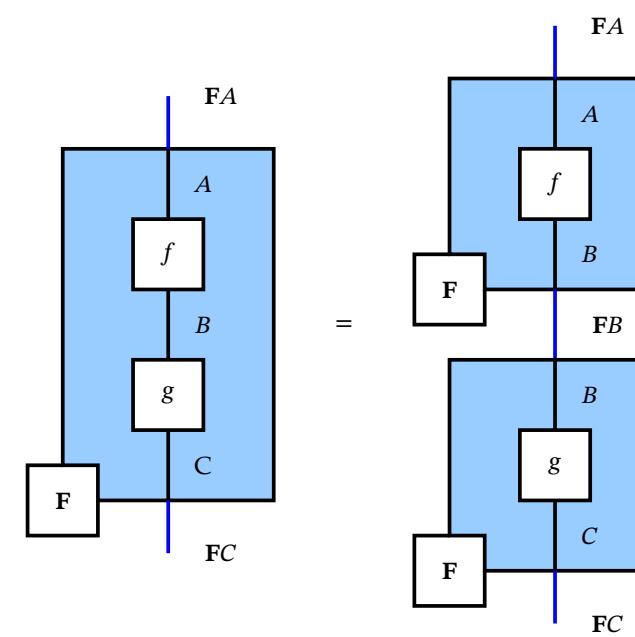


Figure 2.14: Assume that \mathbf{F} is strict monoidal; without loss of generality by the strictification theorem, this lets us gloss over the associators and unitors and treat them as equalities. For \mathbf{F} to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_{\mathcal{C}} = I_{\mathcal{D}}$ and $\mathbf{F}A \otimes_{\mathcal{D}} \mathbf{F}B = \mathbf{F}(A \otimes_{\mathcal{C}} B)$. Diagrammatically these structural constraints amount to these equations.

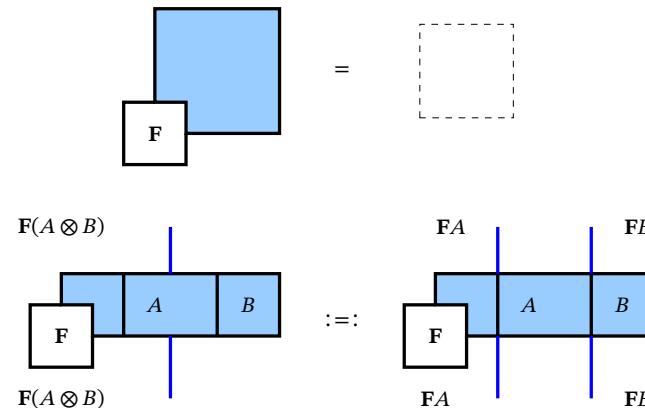


Figure 2.15: What remains is the monoidality of \mathbf{F} , which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes' way of respecting sequential composition, splitting horizontally is how they respect parallel composition.

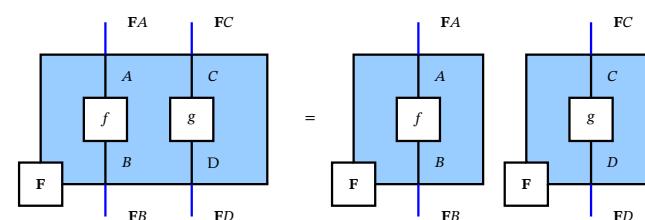


Figure 2.16: And for when we want \mathbf{F} to be a (strict) symmetric monoidal functor, we are just asking that boxes and twists do not get stuck on one another.

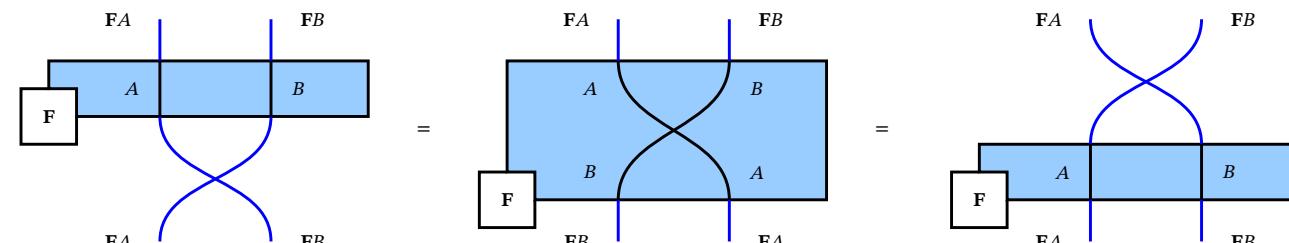
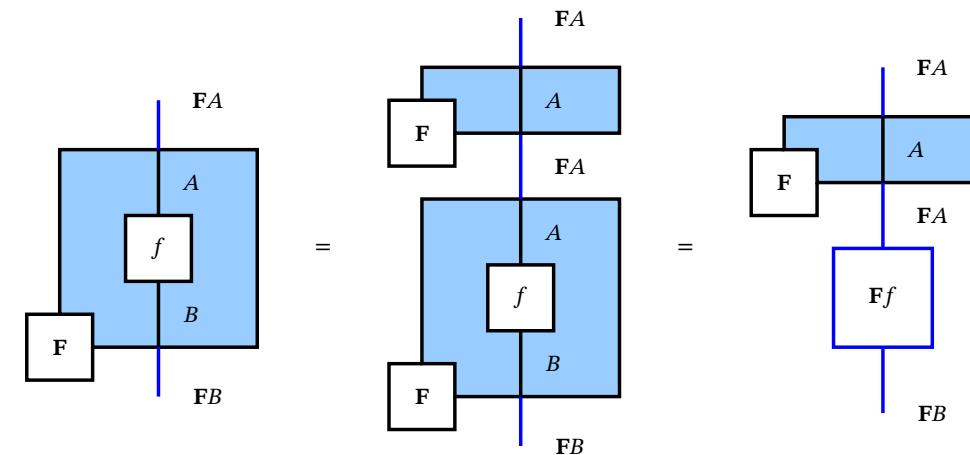


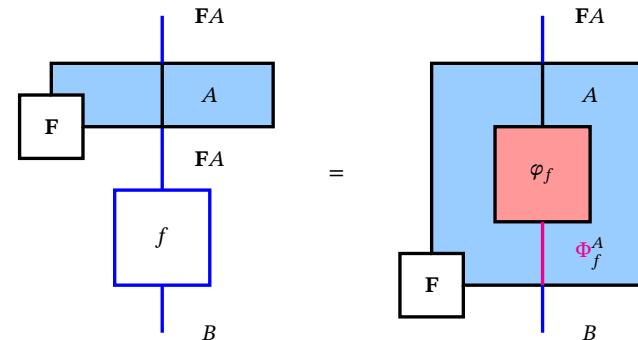
Figure 2.17: To motivate opfibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a functor box out of the bottom. When can we do the reverse? That is, take a morphism in \mathcal{D} and *slide it 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 opfibration.



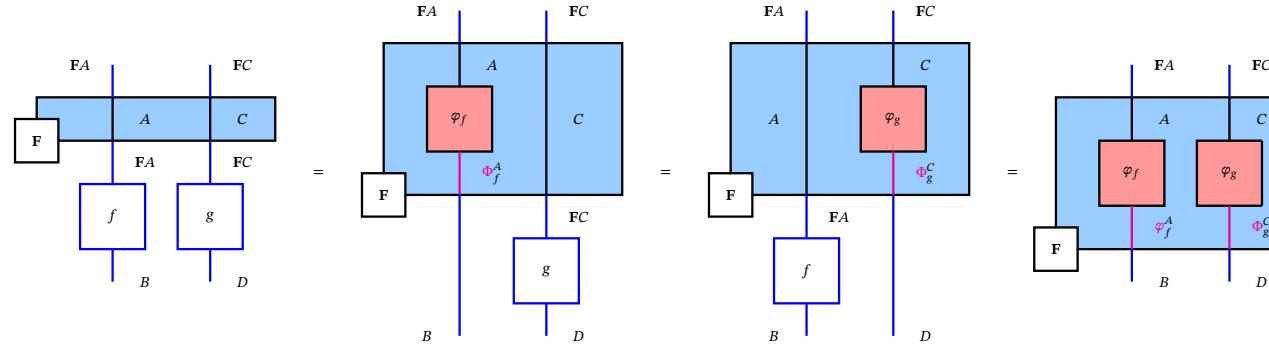
Definition 2.2.2 (Discrete opfibration). $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$ is a *discrete opfibration* when: for all morphisms $f : \mathbf{F}A \rightarrow B$ in \mathcal{D} with domain in the image of \mathbf{F} , there exists a unique object Φ_f^A such that $\mathbf{F}\Phi_f^A = B$ and a unique morphism $\phi_f : A \rightarrow \Phi_f^A$ in \mathcal{C} , such that $f = \mathbf{F}\phi_f$. Diagrammatically, we can present all of the above as an equation reminiscent of sliding a morphism *into* a functor box from below. The process inside the box is called *the lift* of the process that was slid in. The collection of all lifts over a wire or box is called *the fibre over* that wire or box.

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

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



Definition 2.2.3 (Monoidal discrete opfibration). We consider \mathbf{F} to be a (*strict, symmetric*) monoidal discrete opfibration when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the depicted equations relating lifts to interchange hold. The diagrammatic motivation for the additional coherence equations is that – if we view the lifts of opfibrations as sliding morphisms into functor boxes – we do not want the order in which sliding occurs to affect the final result. In this way, lifts behave as graphical primitives in the same manner as interchange isotopies and symmetry twists.



Postscript: Stefano Gogioso observes that this property may be simplified to asking that for all $f : \mathbf{F}A \rightarrow B$, there exists a unique $\varphi_f : A \rightarrow \Phi_f$ such that for all objects C , $\mathbf{F}(\varphi_f \otimes 1_C) = f \otimes 1_{\mathbf{F}C}$, which is potentially easier to verify, and moreover related to "complete" variants of properties (c.f. complete positivity in categorical quantum mechanics) where a property is stable under tensors with the identity. After helpful discussions with Joe Moeller, in joint and ongoing work with Caterina Puca analysing functor boxes we show that these equations are equivalent to asking that the cartesian lift of tensors is equal to the tensor of cartesian lifts. This condition in conjunction with asking for the underlying functor to be strict monoidal and a opfibration then agrees with the "canon" definition of monoidal opfibrations given by Mike Shulman, for the discrete case.

2.2.1 What are they good for?

Figure 2.18: Now we try to use monoidal discrete opfibrations to help us solve the speaker's non-functoriality problem (Example 2.1.3). First we flip over the labels and introduction rules for adverbs. Call this a *dependent CFG*, or *dCFG*. There are several ways to do this formally, by e.g. specifying a new string-diagram signature from the old one or assuming rigid autonomous completion, and it doesn't matter which we use.

Figure 2.19: Treating the label as a test rather than a state will allow the opfibration-box to choose the right version based on the domain wires as it expands top-down. In this case, since CFGs are planar, flipping causes no confusion, since we can always flip the labels back over. Recall that opfibrations can decide which lift to depict given a choice of codomain wires. We would like to encode the dependency of the upside-down adverb labels and introduction rules as lifts that depend on the lift of the verb wire, which may be either an intransitive or transitive verb.

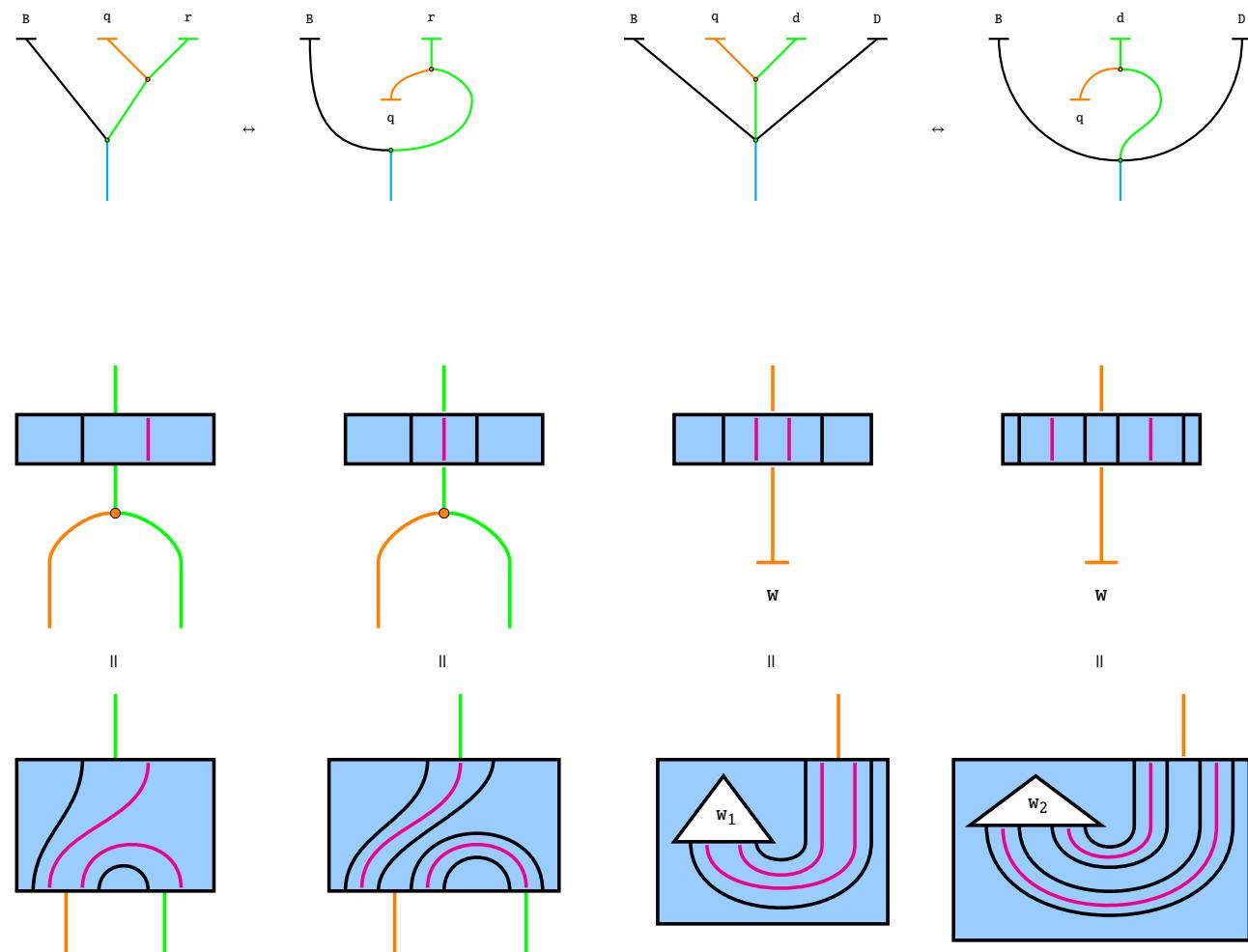


Figure 2.20: Instead of *us* making the choice, we can force the choice using the information of the CFG structure. Starting from a dCFG diagram, the state-labels have unique lifts: noun labels in CFGs correspond uniquely to noun-states in pregroup diagrams, and verb labels to verb-states which may be either intransitive or transitive. This obtains the first equation. The second equation is obtained by monoidality. The third "eating downwards" equation is obtained by the opfibration property; note that because the codomain wires before the lift are already decided to be those of an intransitive verb's pregroup type, the correct adverb introduction rule can be selected for the lift.

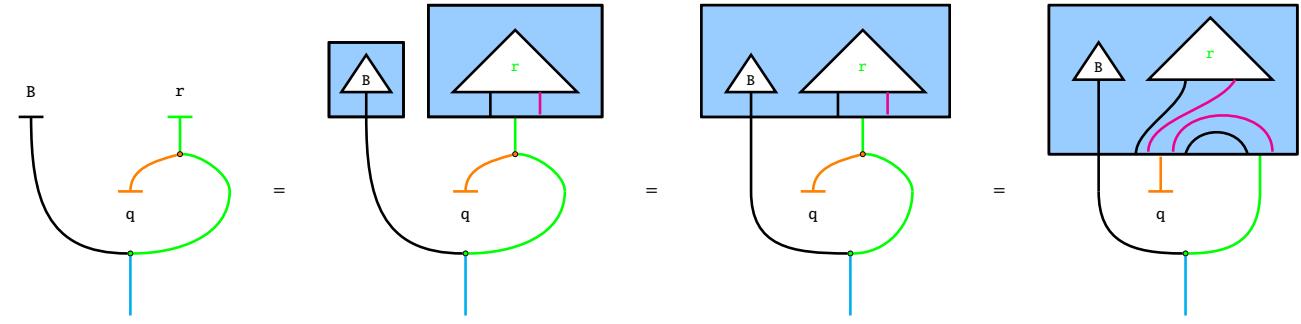
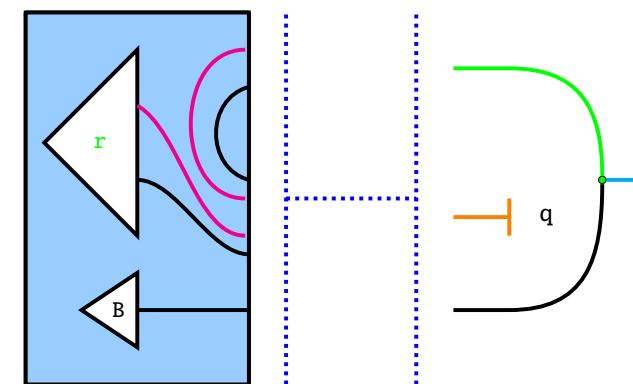


Figure 2.21: But there's a technical problem. We have been assigning wires from the codomain of the lift to the dCFG implicitly, by grouping wires together visually to indicate which wires inside the functor box correspond to wires outside. However, when we consider the algebraic data available, all we know is depicted in the figure: we need some way to assign the wires. Solving the wire assignment problem will be the focus of the next section.

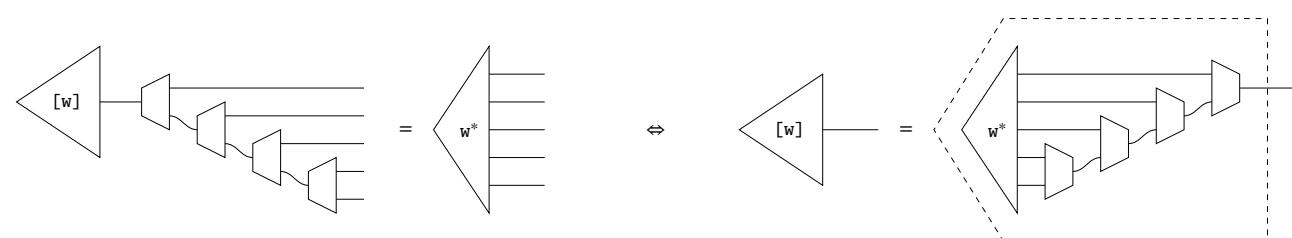


2.3 Strictified diagrams for monoidal categories

The crux of the issue sketched in Figure 2.21 is that while pregroup *proofs* – viewed as sequent trees – syntactically distinguish the roots of subtrees, interpretation as pregroup *diagrams* in a monoidal category forgets the subtree structure of the specific proof the diagram arises from. But it is precisely this forgotten structure that contains the algebraic data we require to keep track of (co)domain data diagrammatically. So a solution would be to force the diagrams in the blue domain recording pregroup data to hold onto this proof structure. For this purpose we use strictified diagrams for monoidal categories, defined in the margins.

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

Construction 2.3.3 (Pregroups with bracketing). Let \mathbf{PGD} be the rigid monoidal category generated by pregroup states and (directed) cups. We define pregroups-with-bracketing as a category denoted $\overline{\mathbf{PGD}}^*$, which is obtained as follows. Throughout, denote the rigid monoidal tensor product as \otimes and the strictified tensor as \bullet . For each pregroup state $w : I_\otimes \rightarrow x_1 \otimes \cdots \otimes x_n$ that generates \mathbf{PGD} , we create two corresponding generators $w^* : I_\bullet \rightarrow x_1 \bullet \cdots \bullet x_n$ and $[w] : I_\bullet \rightarrow (\cdots (x_1 \bullet x_2) \bullet x_3) \cdots \bullet x_n$. $[w]$ is a left-bracketed tensoring, and w^* is fully detensored. Note that $[w]$ and w^* coincide for words typed with singletons. We ask for the following family of relations: either the left or the right implies the other in the presence of equations governing the structural isomorphisms.



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

1. All $[w]$
2. Let $[A \cdot B \cdots Z]$ denote the left-nested tensoring $((A \otimes B) \cdots \otimes Z)$, and let \mathbf{X} denote $(\bigotimes_i X_i)$. For each directed cap $\mathbf{X} \otimes \mathbf{X}^{-1} \rightarrow I$ (and symmetrically for caps of the other direction and cups), and for each pair of brack-

Postscript: it was overkill to use nonstrict diagram machinery for this, and everything could have also been done directly with lists of objects, but I am too lazy to redo it all.

Definition 2.3.1 (Strictified string diagrams). (Presentation from [WGZ22]) Fix an arbitrary (non-strict) monoidal category \mathcal{C} . The *strictification* $(\overline{\mathcal{C}}, \bullet)$ is defined as follows (where strictness of $\overline{\mathcal{C}}$ entitles use of string-diagrammatic notation):

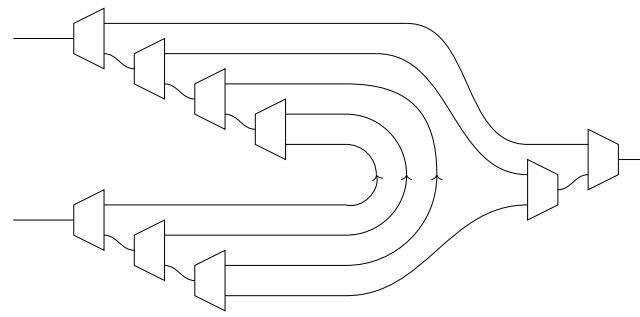
- (1) Objects \overline{A} for each $A \in \mathcal{C}$
- (2) The following generators, with $\overline{f} : \overline{A} \rightarrow \overline{B}$ for each $f \in \mathcal{C}(A, B)$, where we adopt the convention of notating the monoidal unit with a dashed line:

$$\begin{array}{ccc} \overline{A} & \xrightarrow{\Phi} & \overline{A \otimes B} \\ \overline{B} & \xrightarrow{\Phi^*} & \overline{A} \\ \swarrow \varphi & & \searrow \varphi^* \\ I_{\mathcal{C}} & \xrightarrow{\overline{f}} & I_{\mathcal{C}} \end{array}$$

- (3) The following functoriality equations:

$$\begin{array}{ccc} \overline{A} & \xrightarrow{[1_A]} & \overline{A} & = & \overline{A} & \xrightarrow{\quad} & \overline{A} \\ \overline{f} & \xrightarrow{\quad} & \overline{g} & = & \overline{f} & \xrightarrow{\quad} & \overline{f; g} \end{array}$$

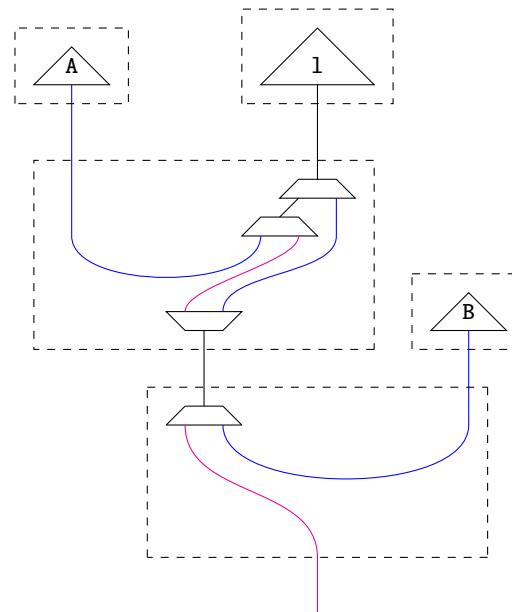
eted types $[A \cdot X]$ and $[X^{-1} \cdot B]$, we ask for a generator that detensors, applies the directed cup on S , and then retensors while respecting the bracketing structure of A and B to obtain $[A \cdot B]$. Diagrammatically this amounts to asking for generators that look like the following, that mimick a single proof step.



Example 2.3.4 (Pregroups with bracketing recover proof trees). The essence of the construction is to maintain a correspondence with proof-tree structure: a left-bracketed collection of types corresponds to a pregroup typing that is stuck together as the outcome of a sequent rule and must thereafter travel together. Starting from bracketed word states $[w]$, point 2 of the construction maintains an invariant correspondence that bracketed collections of types are the roots of proof steps.

$$\frac{\text{Alice : } n \quad \text{likes : } -n \cdot s \cdot n^-}{\text{Alice_likes : } s \cdot n^-} \quad \text{Bob : } n$$

$$\frac{}{\text{Alice_likes_Bob : } s}$$



(4) The following adapter equations:

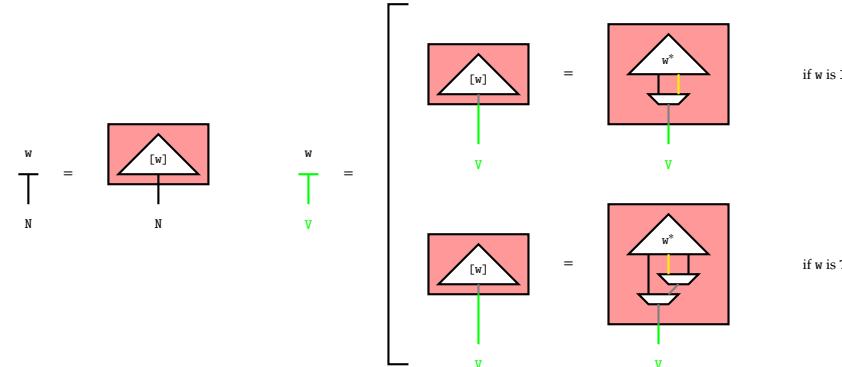
$$\begin{array}{c} \Phi^* \xrightarrow{\quad} \boxed{\bar{f}} \xrightarrow{\quad} \Phi \\ \downarrow \quad \downarrow \\ \Phi \xrightarrow{\quad} \boxed{f \otimes g} \xrightarrow{\quad} \Phi^* \end{array} = \begin{array}{c} \Phi^* \xrightarrow{\quad} \boxed{\bar{f} \otimes g} \xrightarrow{\quad} \Phi \\ \downarrow \quad \downarrow \\ \Phi \xrightarrow{\quad} \boxed{\bar{f}} \xrightarrow{\quad} \Phi^* \end{array}$$

$$\begin{array}{c} \langle \varphi \rangle \xrightarrow{\quad} \varphi^* \\ \downarrow \quad \downarrow \\ \varphi^* \xrightarrow{\quad} \varphi \xrightarrow{\quad} \dots \end{array} = \begin{array}{c} \langle \varphi \rangle \xrightarrow{\quad} \dots \\ \downarrow \quad \downarrow \\ \dots \end{array}$$

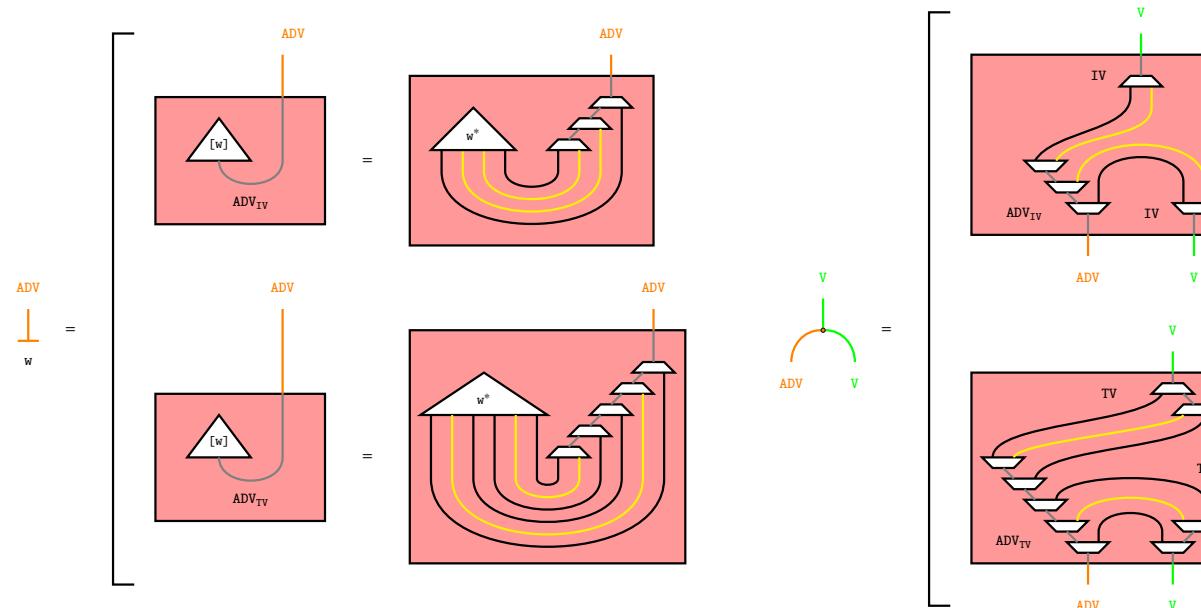
(3) The following representations of the natural isomorphisms in the definition of a monoidal category:

$$\begin{array}{l} \bar{\alpha} := \Phi^* \xrightarrow{\quad} \boxed{\Phi^*} \xrightarrow{\quad} \Phi \xrightarrow{\quad} \Phi \\ \bar{\alpha}^{-1} := \Phi^* \xrightarrow{\quad} \boxed{\Phi^*} \xrightarrow{\quad} \Phi \xrightarrow{\quad} \Phi \\ \bar{\lambda} := \Phi^* \xrightarrow{\quad} \langle \varphi^* \rangle \xrightarrow{\quad} \Phi \\ \bar{\rho} := \Phi^* \xrightarrow{\quad} \langle \varphi \rangle \xrightarrow{\quad} \Phi \\ \bar{\lambda}^{-1} := \langle \varphi \rangle \xrightarrow{\quad} \Phi \xrightarrow{\quad} \Phi \\ \bar{\rho}^{-1} := \langle \varphi \rangle \xrightarrow{\quad} \Phi \xrightarrow{\quad} \Phi \end{array}$$

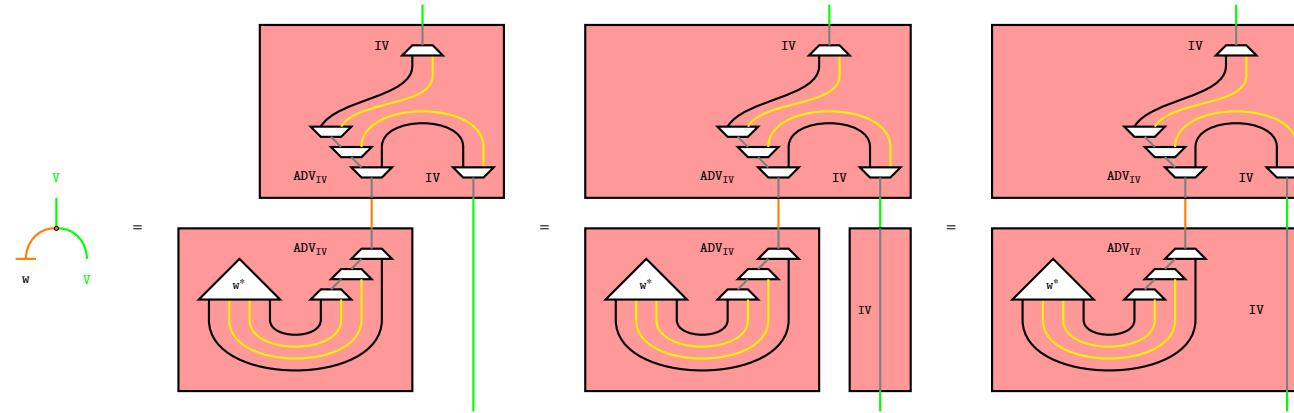
Construction 2.3.5 (Discrete monoidal opfibration from pregroups with bracketing into dependent CFGs). We aim to elucidate the pregroup with bracketing in sufficient detail to describe the functor into dCFGs; in particular, we need to know what the generators of the pregroup are. The fibre over a noun state in the dCFG is the corresponding noun-state in the bracketed pregroup. The fibre over a verb state in the dCFG is either an intransitive or transitive word-state, depending on the word w . Note that only the $[w]$ are available as generators, through we may reason about them as if they are tensor-bracketings of w^* .



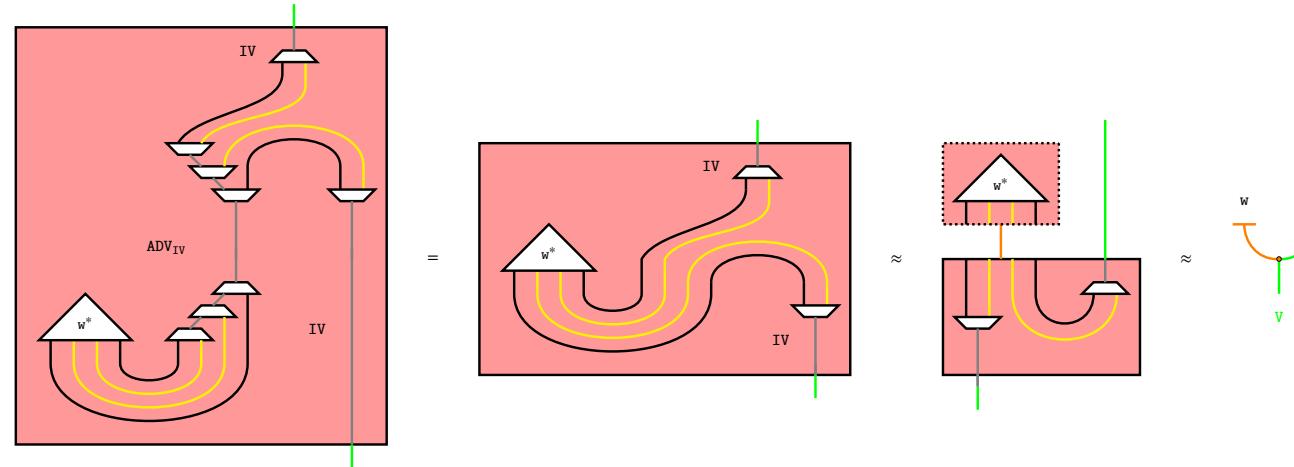
We depict the case of adverbs explicitly. Here are two lifts of the adverb label in the fibre of the opfibration corresponding to the intransitive and transitive case, and the corresponding lifts for the introduction rule for adverbs.



The correspondence of introduction rules in the dCFG to proof steps in the sequent formulation of pregroup proofs (c.f. Example 2.3.4) is obtained obliquely, because labels are facing the wrong way; hence the cups for the lifts of labels. We can observe the correspondence by the following diagrams. The second equation is a supplied choice of lift on the V wire, so that the third monoidality equation allows the top and bottom boxes to typematch.



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



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

$$S, N, V, ADV, ADP$$

The determined equations of the system are the assignments of the types N and S .

$$S = s$$

$$N = n$$

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

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

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

$$V_1 = (-n \cdot s)$$

$$V_2 = (-n \cdot s \cdot n^-)$$

The three dependent types are:

$$V_{x \mapsto 1(x)} = (-n \cdot s)$$

$$ADV_x = V_x \cdot V_x^-$$

$$ADP_x = -V_x \cdot V_{(x)1} \cdot n^-$$

The types V, ADV, ADP are hence indexed over a string language $x := 1 \mid 2 \mid 1(x) \mid (x)1$. We have depicted the solutions for ADV_1 and ADV_2 . The rest are obtainable inductively, where the bracketing structure is handled by the tensors and detensors of the strictified pregroup diagrams. The pregroup typing solutions for V, ADV, ADP across indices are unique, as the latter two generators of the string language correspond to verbs with sentential complement and adpositions respectively, so by the bracketing structure, indices correspond uniquely to dCFG diagrams up to labels. The family of pregroup typing solutions yield the required generators, which we use to populate the fibres over the three dependent type labels and their introduction rules.

Proposition 2.3.6 (Construction 2.3.5 is a discrete monoidal opfibration). *Proof.* Monoidality is evident. For unique lifts, we observe that for each bracketing of wires, construction 2.3.3 guarantees a unique lift for each introduction rule in the dCFG, and Construction 2.3.5 guarantees a unique lift for each label. For the additional interchange condition of Definition 2.2.3, it suffices to observe that the introduction rules of dependent labels uniquely determine the codomain of the lift given the domain, and that by design in Construction 2.3.5, independent labels as states have predetermined lifts. \square

2.4 Monoidal cofunctor boxes

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

Definition 2.4.3 (Bijective-on-objects functor). [Defn 2.8]. A functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is *bijective-on-objects* if for all $d \in \mathcal{D}$, there exists a unique $c \in \mathcal{C}$ such that $Fc = d$.

Proposition 2.4.4 (Cofunctors as spans of functors). [Prop 2.10] all cofunctors $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ correspond to spans of functors where the left leg L is bijective on objects and the right leg R is a discrete opfibration:

$$\begin{array}{ccc} & \Lambda(f, \varphi) & \\ L \swarrow & & \searrow R \\ \mathcal{C} & & \mathcal{D} \end{array}$$

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

$$\begin{array}{ccc} & \mathcal{X} & \\ F \swarrow & & \searrow G \\ \mathcal{C} & & \mathcal{D} \end{array}$$

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

Definition 2.4.5 (Monoidal cofunctor). A monoidal cofunctor, following Proposition 2.4.4, is a span of functors such that the left leg is monoidal and bijective-on-objects, and the right is a monoidal discrete opfibration by Definition 2.2.3.

Construction 2.4.6 (Monoidal cofunctor box). A monoidal cofunctor box first uses the inside-out convention for functor boxes for the right leg, and then the outside-in convention for the left leg.

Proposition 2.3.2 ($\tilde{\mathcal{M}}$ and \mathcal{M} are monoidally equivalent). [WGZ22]

These definitions and conventions follow [Cla23]. Given a (small) category \mathcal{C} we notate the objects \mathcal{C}_0 and the morphisms \mathcal{C}_1 , hence a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ consists of an object assignment $F_0 : \mathcal{C}_0 \rightarrow \mathcal{D}_0$ and a morphism assignment $F_1 : \mathcal{C}_1 \rightarrow \mathcal{D}_1$.

Definition 2.4.1 (Cofunctors). [Defn 2.2]. A *cofunctor* $(f, \varphi) : \mathcal{C} \nrightarrow \mathcal{D}$ consists of a function $f : \mathcal{C}_0 \rightarrow \mathcal{D}_0$ which I'll call *lowering*, together with a *lifting operation* φ , a function that maps pairs of objects of \mathcal{C} and certain morphisms in \mathcal{D} to morphisms of \mathcal{C} :

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

The following conditions are required:

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

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

2. The lifts of identities are identities.

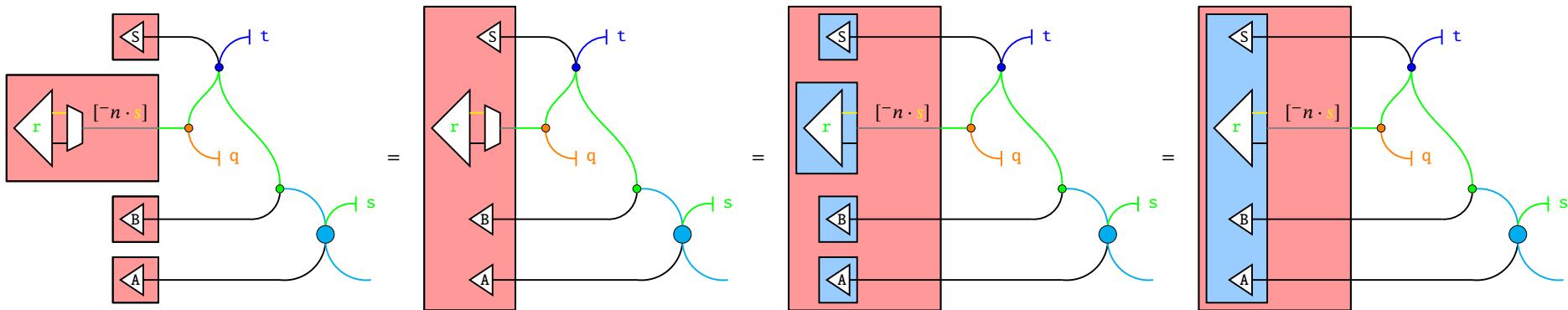
$$\varphi(c, 1_{f(c)}) = 1_c$$

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

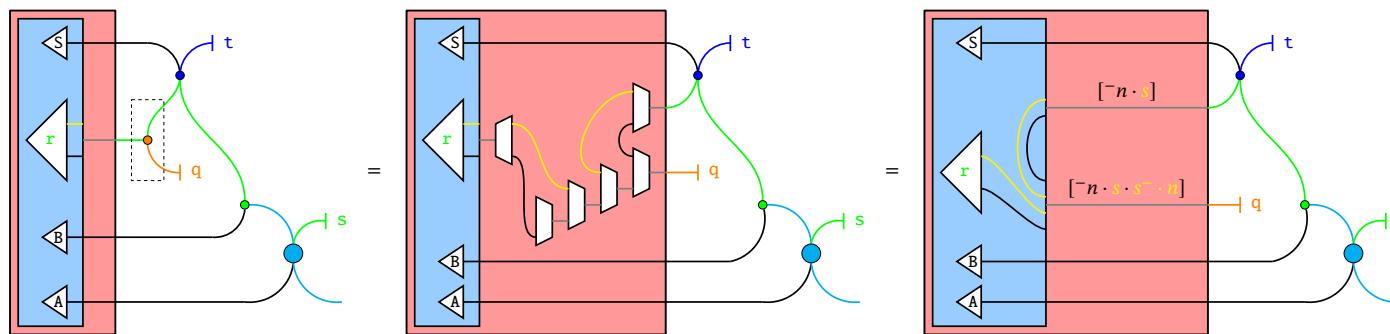
$$\varphi(c, v \circ u) = \varphi(\text{cod}(\varphi(c, u)), v) \circ \varphi(c, u)$$

Remark 2.4.2. Conditions 2 and 3 of the definition of cofunctor are reminiscent of functors. It is instructive but tedious to calculate with the base definition. We use the slick alternative formulation by Bryce Clarke.

Example 2.4.7 (Turning dCFGs into pregroup diagrams with a monoidal cofunctor). Here is a more involved example, which uses a CFG and running example from the next chapter: Alice sees Bob quickly run to school. The apex is given by Construction 2.3.3. The right leg is the monoidal discrete opfibration from Construction 2.3.5 into the dCFG. In the first diagram below, we apply the opfibration to the word states, which have unique lifts. The second diagram follows by monoidality. In the third, we apply the left leg of the cofunctor using the outside-in convention, which is the evidently bijective-on-objects monoidal functor to the ambient rigid autonomous category of pregroup diagrams from the free strictification. In the fourth, we apply the monoidality of the inner functor.

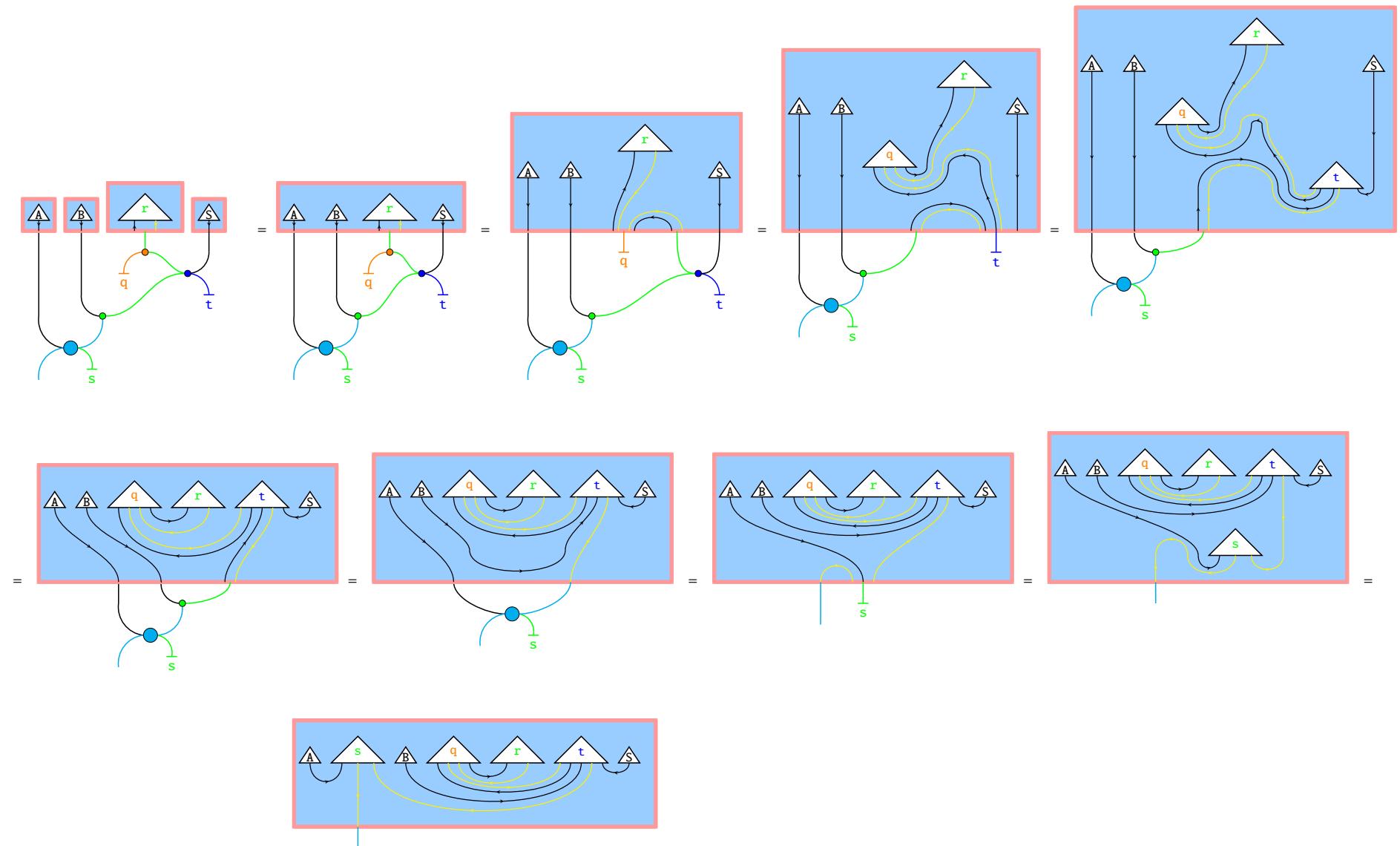


Continuing, for the first equation below, the magenta opfibration box now has already chosen a codomain for the lift, so it can eat the next morphism highlighted in a dashed box. For the second equation, note that eating-rules swap over for the different conventions of functor boxes: while inside-out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



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

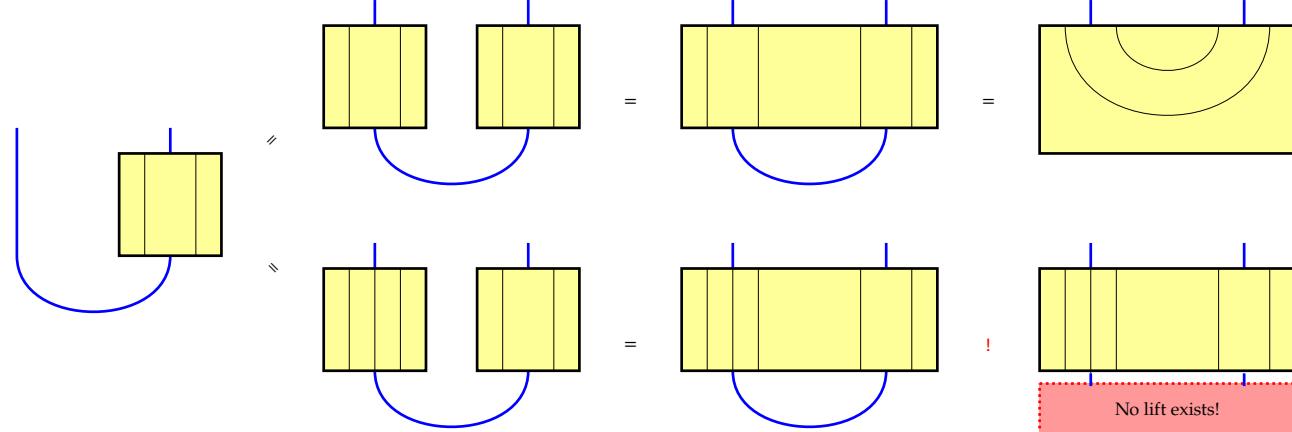
And so we can continue all the way, occasionally straightening out some wires on the inside.



2.5 Monoidal kinda-cofunctor boxes

Now we'd like to use the cofunctor technology we have developed to tackle the other problem, the nonfunctionalit of internal wirings for the listener (Example 2.1.4.) We run into a problem again: the right leg cannot be a discrete opfibration into pregroup diagrams.

Figure 2.22: Starting from the leftmost diagram, in order to let the functor box eat the whole diagram, we need to first choose a lift for the left-sentence wire for the cup. Recalling Figures 2.6 and 2.1.4, there are at least two lifts for the sentence-wire in pregroup diagrams, for the case of two or three noun-wires. Everything works smoothly when the lifts on the two sentence wires of a cup match. When if we make the wrong choice and they don't, there is no lift, because there is no such thing as a cup that has two wires on one end and three on the other. Recall from Definition 2.5.1 that a unique lift is required for *every possible* codomain inside the functor box; so we do not have a discrete opfibration, and so we cannot have a cofunctor.



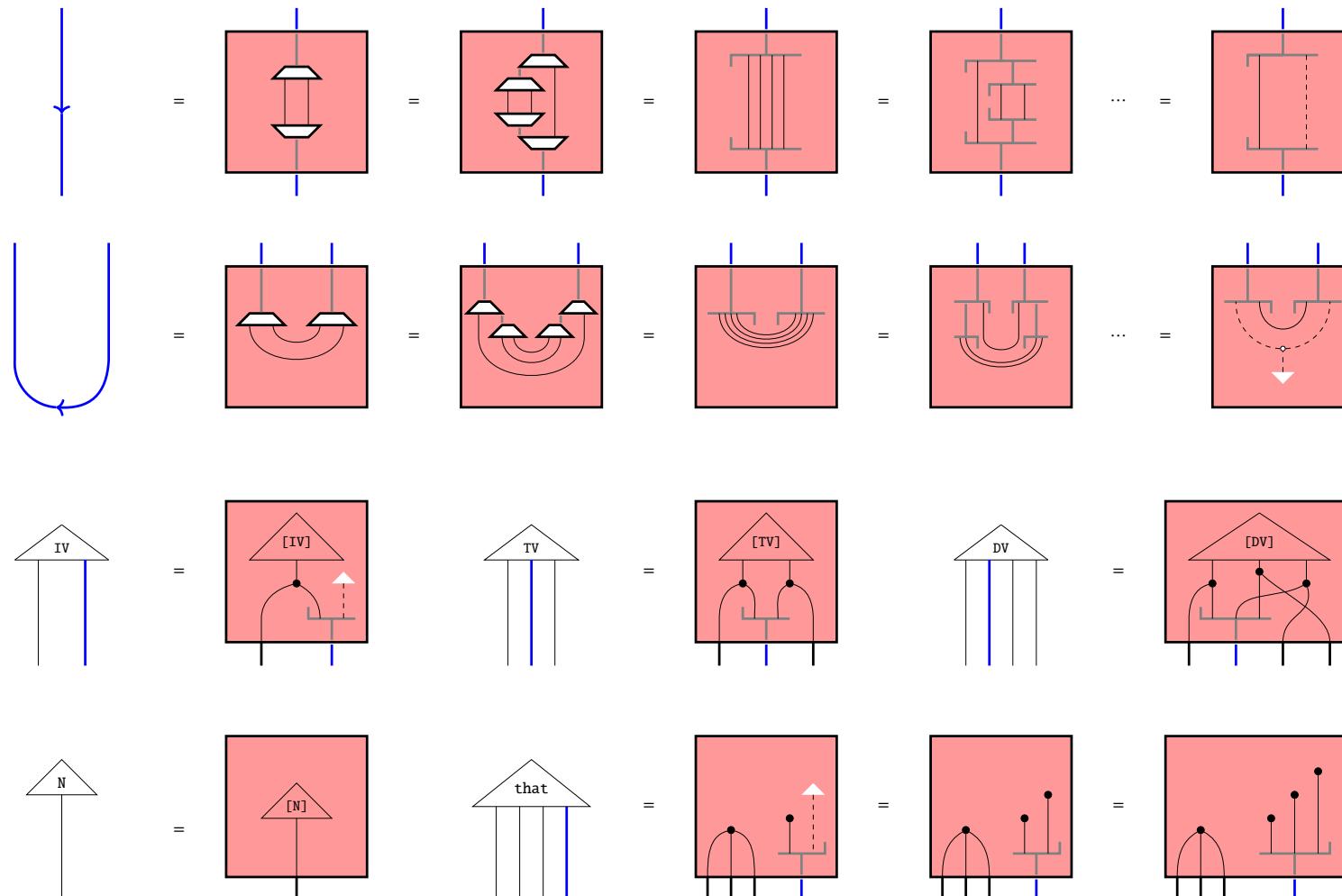
But hold on, if we know that cups need pairs of matching identity-lifts, in the above example it would have been obvious *from the connectivity of the diagram* what the correct choice of lift ought to have been. In order to reason diagrammatically with hungry functor boxes in the way we'd like, we can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

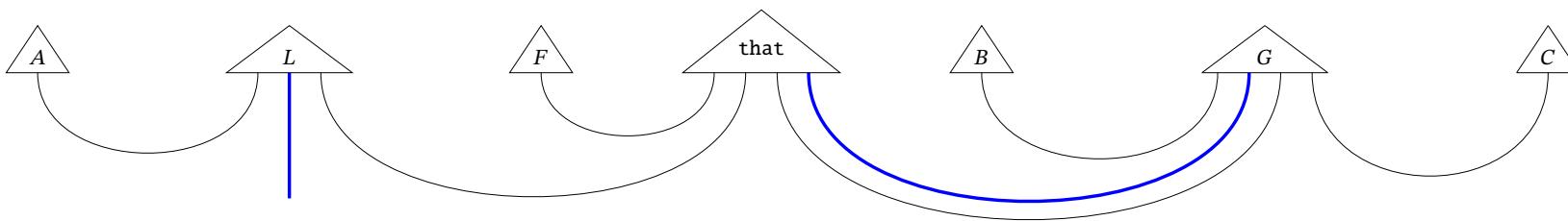
Definition 2.5.1 (Kinda-opfibration). $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$ is a *kinda-opfibration* when for all morphisms $f : \mathbf{F}A \rightarrow B$ in \mathcal{D} with domain in the image of \mathbf{F} , there exists some object Φ_f^A and some $\phi_f : A \rightarrow \Phi_f^A$ in \mathcal{C} , such that $f = \mathbf{F}\phi_f$.

Definition 2.5.2 (Monoidal kinda-opfibration and kinda-cofunctor). Mentally search and replace discrete for kinda- in Definition 2.2.3 and Proposition 2.4.4.

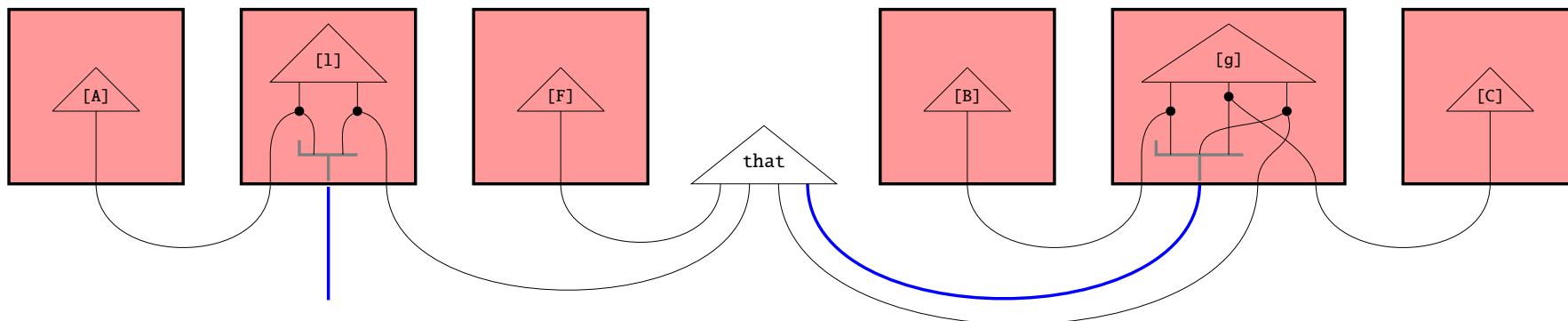
Example 2.5.3 (A monoidal kinda-opfibration into pregroup diagrams from a subcategory of bracketed pregroups with spiders).

Source category is the free strictification of a single wire equipped with a spider, with generator-states implied by the following equations. Target is pregroup diagrams in n, s . Sentence type is bracketings of nouns: alternate bracketing notation is introduced for brevity. Strictified unit is used to distinguish the bracketed single noun wire from the plain noun wire, which is its own lift. Bracketing monoidal units gives the Dyck language, which may be in principle used to distinguish turning numbers in the rigid autonomous setting, omitted for brevity. For more internal wirings for other grammatical categories, see [WC]. This data suffices to model how the listener recovers the data conveyed by the speaker, c.f. Example 2.1.4. Solution to Example 2.1.4 on next page.

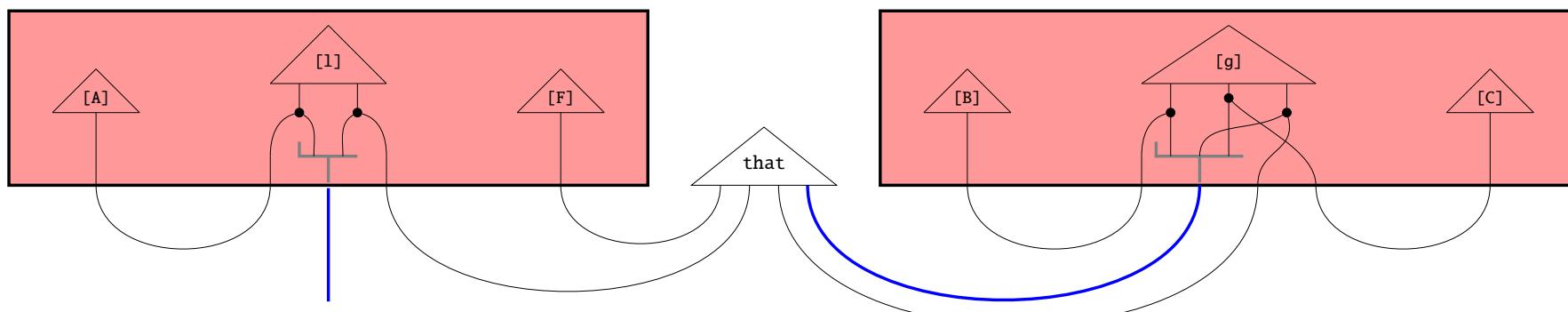




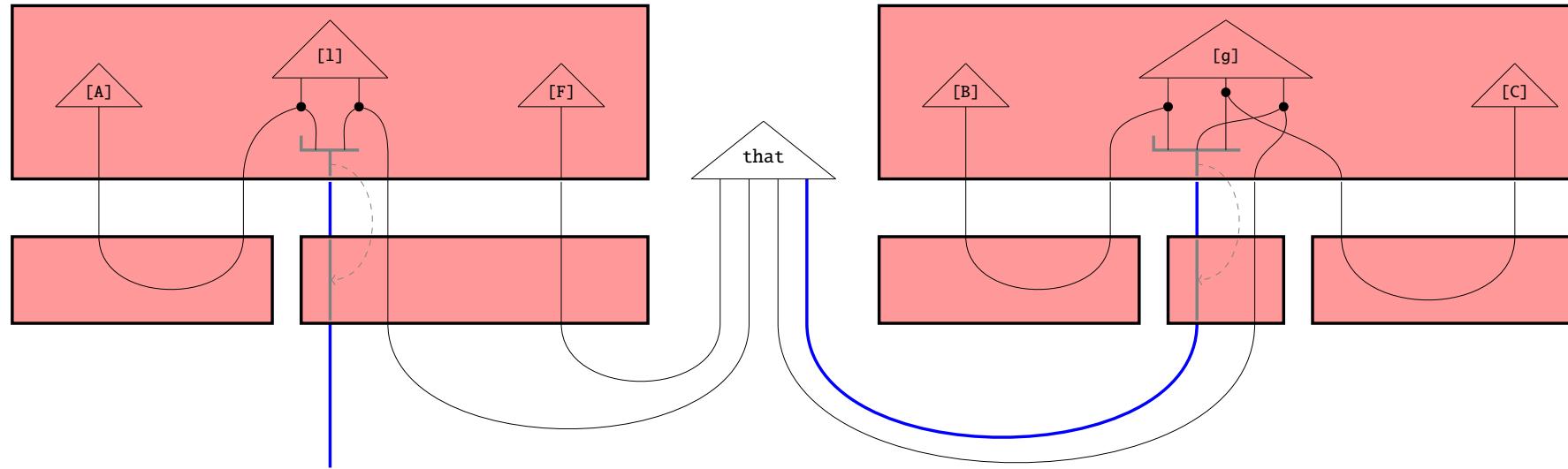
Some lifts are determined.



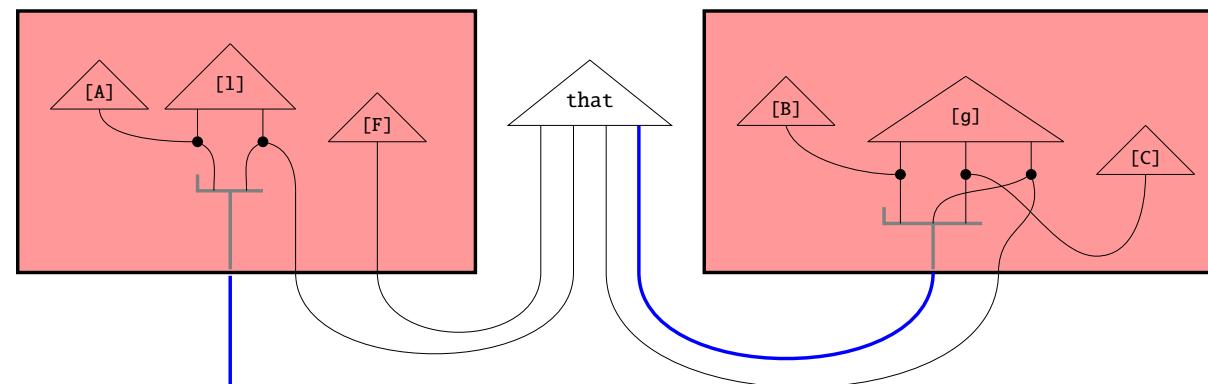
Merge.



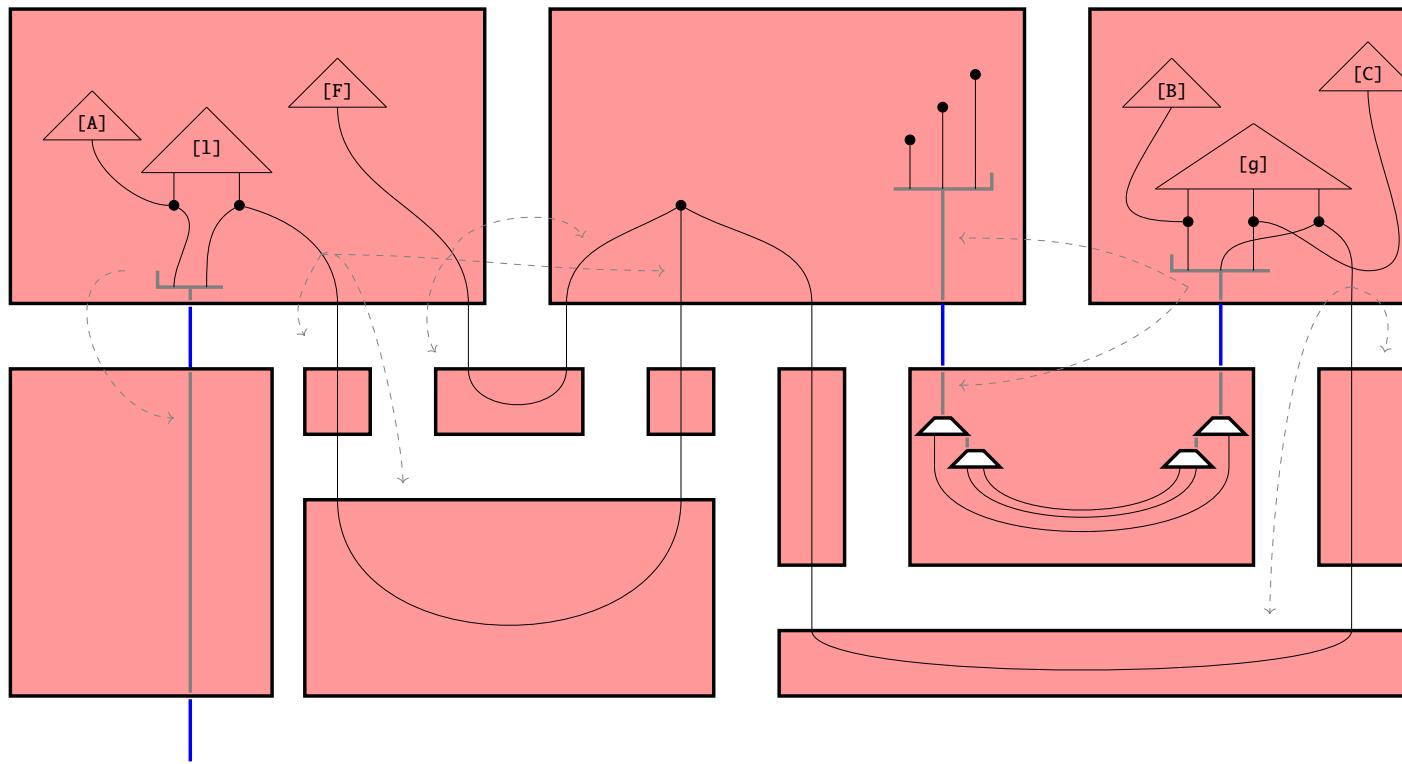
Noun-cups have determined lifts. Typematching lifts inferred from connectivity.



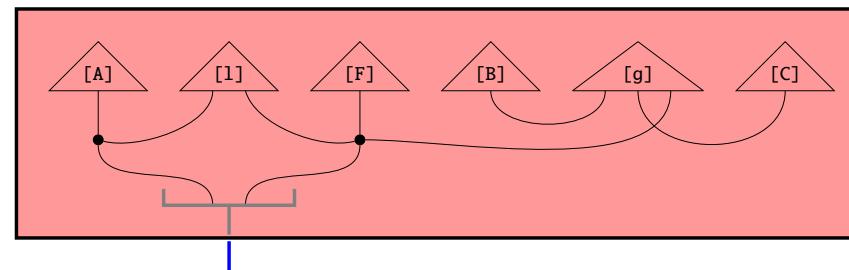
Merge.



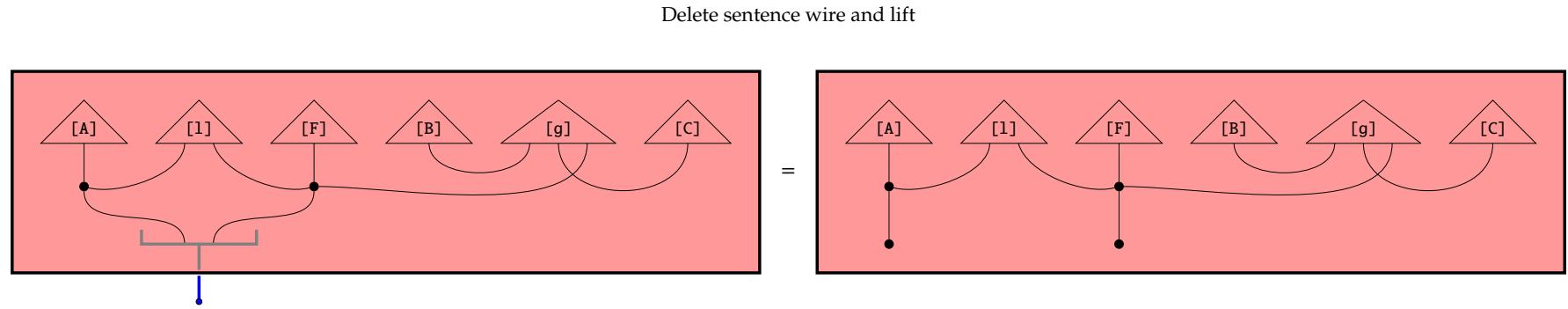
Type-matching lifts inferred from connectivity, and type-restriction of cups.



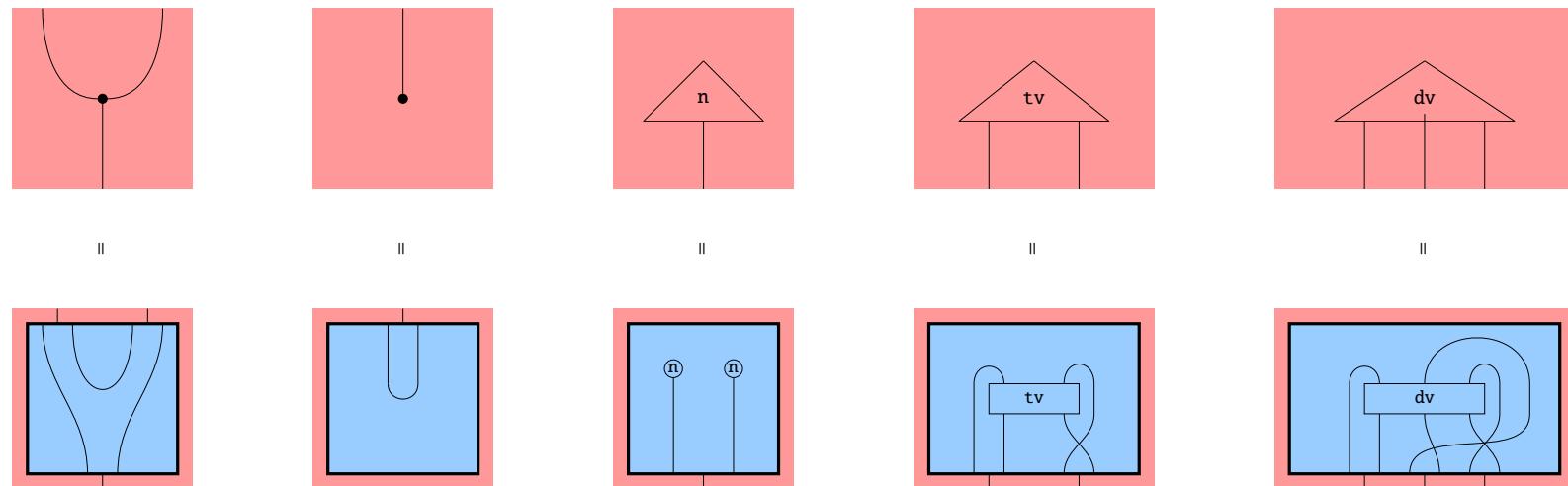
Merge. Cancel brackets. Simplify spiders.



Example 2.5.4 (...and from there to text circuits). Recall from Example 2.7 that pregroup diagrams can be daisy-chained for text. We'll assume that a finished text is one where the sentence-wire is deleted (i.e. no more sentences to chain.)

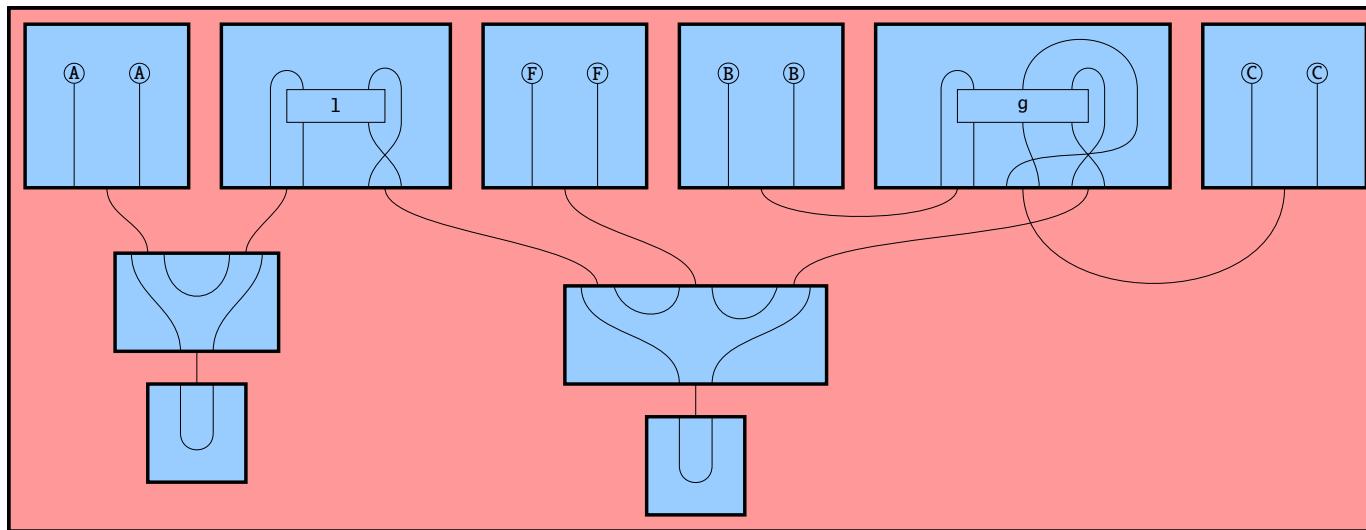


And we'll specify the following monoidal functor from the category we just lifted to in Example 2.5.3 into a strongly compact closed category; preferably one that is a variant of **Prof**, in which one can reason with open string diagrams [Hu, Rom21]. The functor interprets spiders as pairs of pants, and word-states as interesting-looking things. We'll show just the lifts of the monoid part of the spider; the comonoid part is the same thing upside down. A technicality we gloss over is the loss of commutativity of pants, which can be handled by dropping commutativity syntactically from the spiders in the source.

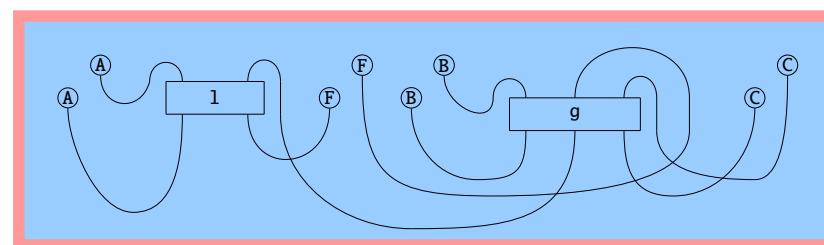


Applying this functor establishes that the same kind of span of functors that gets us from productive to parsing grammars is also sufficient to get us from parsing grammars to text circuits. By mathematically analysing and taking away the bureaucracy of syntax, it becomes evident that the natural habitat of language is not a line, but something else.

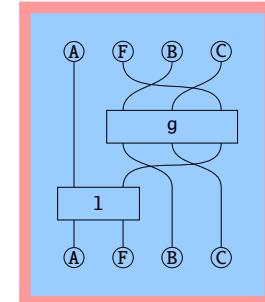
Inner functor.



Merge.



...and simplify:



2.6 Discussion and Limitations

OBJECTION: A HANDFUL OF GADGETS AND EXAMPLES ISN'T A THEORY.

I wanted to understand the nature of communication from first principles. I could not find a satisfying account that reflected the computational constraints of speaking and listening, so I used what mathematics I knew to build a model myself. So even if it is not a theory, I consider it better than nothing at all.

The gadgets may be worthwhile beyond the story I've told here. I'll remark that the monoidal cofunctor technology seems to be just the kind of diagrammatic mathematics required by the "merge"-operation in minimalist syntax. They are probably better off named "merge-boxes".

The examples were necessary to display the mathematics, but this specificity was also necessary in a broader sense. A systematic analysis of communication requires intimacy with specific grammars and a specific semantics, not *formats* of grammar like "All CFGs". Specific grammars that model natural languages, even poorly, are the only relevant objects of study for any form of language intended to communicate information. Once one has a specific grammar that produces sentences in natural language, then to explain communication, one must supply a specific partnered parsing grammar such that on the produced sentences, both grammars yield the same semantic objects by a Montagovian approach, broadly construed as a homomorphism from syntax to semantics. On this account, syntax does not hold a dictatorship over semantics; there are duarchies, and in these duumvirates the two syntaxes and the semantics mutually constrain one another.

HOW FAR DOES THIS APPROACH GENERALISE? WHERE DOES IT APPLY?

Internal wirings are a way to encode relationships between two different kinds of grammar, and a string-diagrammatic semantics. They arise as part of cofunctor boxes that witness a systematic correspondence between two grammars. While we have encoded a specific kind of grammatical resolution choice in the fibres of our lifting-functor, there are easy examples of other kinds of choices a listener has to make in order to parse a sentence that the current theory is not yet elaborated enough to accommodate.

Example 2.6.1 (Garden path sentences). Whereas we assume that grammatical types have already been chosen, 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. 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:

$$\begin{array}{c}
 \frac{\text{the} : n \cdot n^{-1} \quad \text{old} : n \cdot n^{-1}}{\text{the_old} : n \cdot n^{-1}} \quad \text{man} : n \quad \frac{\text{the} : n \cdot n^{-1} \quad \text{boat} : n}{\text{the_boat} : n} \\
 \text{the_old_man} : n \qquad \qquad \qquad \text{the_boat} : n \\
 \text{Not a sentence!}
 \end{array}$$

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:

$$\begin{array}{c}
 \frac{\text{the} : n \cdot n^{-1} \quad \text{old} : n}{\text{the_old} : n} \quad \text{man} : n^{-1} n \cdot s \cdot n^{-1} \quad \frac{\text{the} : n \cdot n^{-1} \quad \text{boat} : n}{\text{the_old} : n} \\
 \text{the_old_man} : s \cdot n^{-1} \qquad \qquad \qquad \text{the_old} : n \\
 \text{the_old_man_the_boat_} : s
 \end{array}$$

Example 2.6.2 (Ambiguous scoping). The uniqueness condition of lifts is too stringent. Consider the following sentence:

Everyone loves someone

The sentence is secretly (at least) two, corresponding to two possible parses. The usual reading is (glossed) $\forall x \exists y : \text{loves}(x, y)$. The odd reading is $\exists y \forall x : \text{loves}(x, y)$: a situation where there is a single person loved by everyone.

WHAT ARE YOUR ASSUMPTIONS? WHAT ARE THEIR LIMITATIONS? In my view, the preceding analysis is fair if one entertains the following three commitments.

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

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

LIMITATIONS OF THE FIRST ASSUMPTION:

The first assumption establishes an idealised view of communication and compositionality where there are no extraneous rules in the language, i.e. that a particular phrase of five or sixty-seven words is to be parsed exceptionally. This is not the case in natural languages, where everyday idioms may be considered semantically atomic despite being compositionally decomposable. For example, in Mandarin, 马上 is the concatenation of "horse" and "up", and would be "on horseback" if interpreted literally, but is treated as an adverbial

"as soon as possible" in imperative contexts. I use the hedging phrase "at some level" in the first assumption to describe the compositionality of semantics just to indicate an assumption that we are not dealing with exceptional rules all the way up.

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

Example 2.6.3 (Exclamations: how meaning affects grammar). The following examples from [lin18] 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 appear to mean similar things.

`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) $\dots \neg \exists x_{Person} \dots$ of `God knows` is lost, and what is left is a literal reading $\dots \neg \text{knows}(\text{God}, \dots) \dots$. 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

LIMITATIONS OF THE SECOND ASSUMPTION:

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

LIMITATIONS OF THE THIRD ASSUMPTION:

The third assumption asks that one entertain string diagrams as representative of what the content of language is, and even so, it still requires some elaboration on what is meant by "understanding", as it is obviously untrue that everyone understands one another. I do not mean understanding in the strong sense as a form of telepathy of mental states – I mean that insofar as the speaker and listener both have their own ideas about cats, sitting, space, and mats, their respective mental models of the **cat sat on the mat** are indistinguishable as far as meaningfully equivalent syntactic re-presentations and probings go; for instance, both speakers ought to agree that the **mat is beneath the cat**, and both speakers ought to agree despite the concrete images in their minds that there is insufficient information to know the colour of the cat from the sentence alone, and so on. This is a shallow form of understanding; consider the case where one communicator is a human with mental models encoded in meat and another is an LLM with tokens encoding who-knows-what – they may be in perfect agreement about rephrasings of texts for an arbitrary finite amount of communication, even if the representations of the latter are not compositional. It would be nice to ask that "mutual understanding" requires structurally equivalent (as opposed to extensionally indistinguishable) meaning-representation mechanisms between language agents c.f. Chomsky's universal grammar, but our means of achieving mutual understanding in practice seems to align with the shallow view: we pose comprehension challenges and ask clarifying questions all at the level of language, without taking a scalpel to the other's head. Depending on one's view of what understanding language entails, it may be that humans and LLMs both understand language in their own way, but mutual understanding between the two kinds is an illusion.

SO WHAT?

Despite these limitations, I believe that this formal approach to grounding relationships between productive and parsing grammars in mathematical considerations surrounding communication has some merits, or at least raises a change in perspective. Theories of grammar by themselves are insufficient to account for communication: a theory of grammar that merely produces correct sentences or correct parses is a 'theory' of language waiting to be or already outperformed in every respect by an LLM. Understanding the nature of syntax as a formal object demands a distinction and reconciliation of speaking and listening grammars. Moreover, semantics must play a role from the start: in the ideal of communication, both speaker and listener have the same semantic information by the end of a single turn, whether that be a logical expression or something else. For these reasons, we may at least conclude that "weak" and "strong" equivalence is just not suited for an adequate account of communication.

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 [BM] only admits the possibility that a speaker using a context-free grammar and a listener using a pregroup grammar *could* understand each other, without providing any explanation *how*. But we already know that users of language *do* understand one another (more-or-less), so weak equivalence is (more-or-less) pointless.

Secondly, "strong equivalence" that seeks equivalence at a structural level between theories of syntax often helps, but is not always necessary. 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 – and has nothing to do with the object being photographed – to demonstrate that all .pngs and .jpeggs are structurally bijective, just as it is overkill to show that, say, context-free grammars are strongly equivalent to pregroup grammars, because there are context-free and pregroup grammars that generate sets of strings that have nothing to do with natural language. It could just as well be that there is a pair of productive and parsing grammar-formats that are not strongly equivalent, but happen to coincide for a particular natural language – in this sense, asking for something like a monoidal cofunctor is a way to check a weaker condition than strong equivalence that achieves the more specific aim of determining whether a pair of productive and parsing grammars for a language are plausibly compatible.

AND WHAT DO YOU KNOW ABOUT FORMAL GRAMMARS? More than I wish I did. See next chapter.

3

Text circuits for syntax

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

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

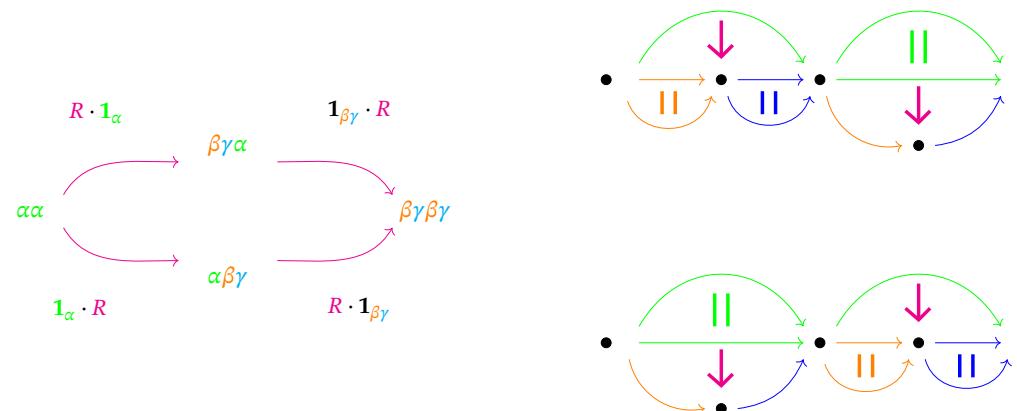
In regular or 1-category theory we are interested in objects up to isomorphism rather than equality - because isomorphic objects are as good as one another. Concretely, two objects X and Y are isomorphic when there exist morphisms $f : X \rightarrow Y$ and $g : Y \rightarrow X$ such that $g \circ f = 1_X$ and $f \circ g = 1_Y$. Lifting this philosophy to n -categories, we can replace equalities $f = g$ of k -morphisms (for $k < n$) with $(k+1)$ -isomorphisms $f \simeq g$. This extends also to the equalities in the definition of a $(k+1)$ -isomorphism, all the way up to dimension n at which point we are content with ordinary equality again. The idea of replacing equality with isomorphism can even extend to the associativity, unitality and interchange laws governing the composition operations. If left to be ordinary equalities, we speak of a *strict n -category*. If only witnessed by a coherent system of isomorphisms, we have a *weak n -category*. Unsurprisingly the weak variant is much harder to formalise, but also much more expressive once $n > 2$.

Mathematicians, computer scientists, and physicists may have good reasons to work with n -categories [Bae97], but what is the value proposition for formal linguists? A practical draw for the formal syntactician is that weak n -categories provide a natural setting to model generic rewriting systems, and that is what we will focus on here. Here, we work with a semi-strict model in which associativity and unitality remain strict, while interchange is weak. This fact is hidden behind a convenient graphical notation, and we do not miss out on any of the expressivity of fully weak higher categories. Additionally, this system has an online proof assistant homotopy.io. For formal details the reader is referred to [nLaa, Dor23, RV19, HRV22]. In this setting we will demonstrate how weak n -categories provide a common setting to formalise string-rewrite and tree-adjoining grammars, setting the stage for us to specify a circuit-adjoining grammar for text circuits later on.

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 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 $\alpha, \beta, \gamma : \star \rightarrow \star$. In this category, every morphism $\star \rightarrow \star$ 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 rigueur* when we consider formal languages to be subsets of Σ^* . She might be happy to declare that these two rewrites are simply equal; categorically this is tantamount to her declaring that any two 2-cells in the 1-object-2-category that share the same source and target are in fact the same; between any such pair of 2-cells there is only one

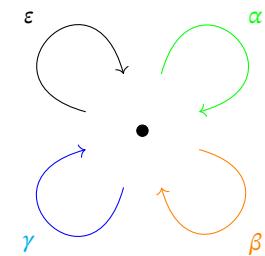


Figure 3.1: The category in question can be visualised as a commutative diagram.

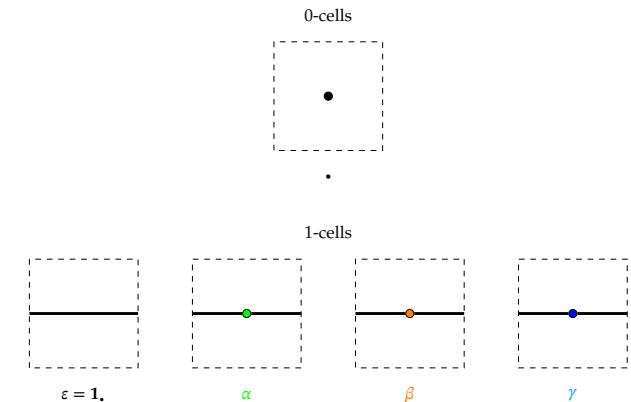


Figure 3.2: When there are too many generating morphisms, we can instead present the same data as a table of n -cells; there is a single 0-cell \star , and three non-identity 1-cells corresponding to α, β, γ , each with source and target 0-cells \star . Typically identity morphisms can be omitted from 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.

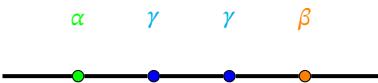


Figure 3.4: The string-diagrammatic view, where \star is treated as a wire and morphisms are treated as boxes or dots is an expression of the same data under the Poincaré dual.

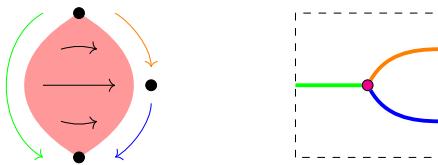
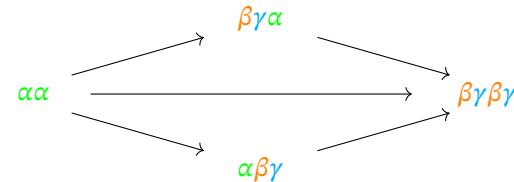
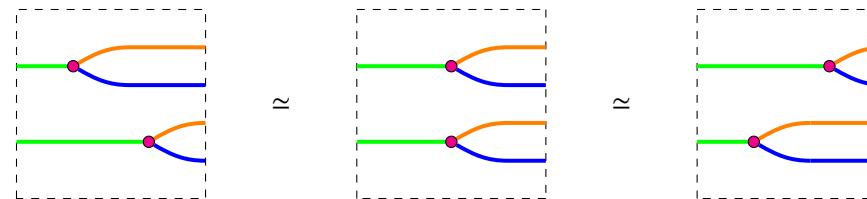


Figure 3.5: We can visualise the rule as a commutative diagram where R is a 2-cell between the source and target 1-cells. Just as 1-cells are arrows between 0-cell points in a commuting diagram, a 2-cell can also be conceptualised as a directed surface from a 1-cell to another. Taking the Poincaré dual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

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



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



In fact, Bob gets to express a new kind of rewrite in the middle: the kind where two non-conflicting rewrites happen *concurrently*. The important aspect of Bob's view over Alice's is that equalities have been replaced by isomorphisms between syntactically unequal rewrites. In this case, the demotion of interchange equalities to isomorphisms means that Bob is dealing with a *weak* 1-object-2-category (in general the demotion of equalities to isomorphisms is not synonymous to weakness.) Bob does have 3-cells that relate different 2-cells with the same source and target, but all of Bob's n -cells for $n \geq 3$ are equalities, rather than isomorphisms.

3.1.2 Tree Adjoining Grammars

Definition 3.1.1 (Elementary Tree Adjoining Grammar: Classic Computer Science style). An elementary TAG is a tuple

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

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

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

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

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

Auxiliary trees satisfy the following constraints:

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

Further, there are two operations to build what are called *derived trees* from elementary and auxiliary trees. *Substitution* replaces a substitution marked leaf node 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 X, X^* , and a derived tree γ at an interior node X of γ . Removing the X node from γ separates it into a parent tree with an X -shaped hole for one of its leaves, and possibly multiple child trees with X -shaped holes for roots. The result of adjoining is obtained by identifying the root of β with the X -context of the parent, and making all the child trees children of β 's foot node X^* .

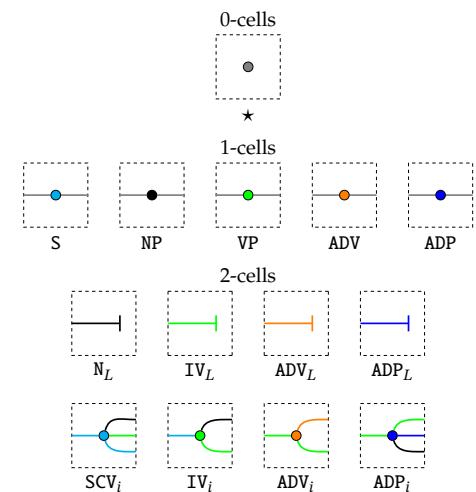


Figure 3.6: We can describe a context-free grammar with the same combinatorial rewriting data that specifies planar string diagrams as we have been illustrating so far. Here is a context-free grammar for Alice sees Bob quickly run to school.

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

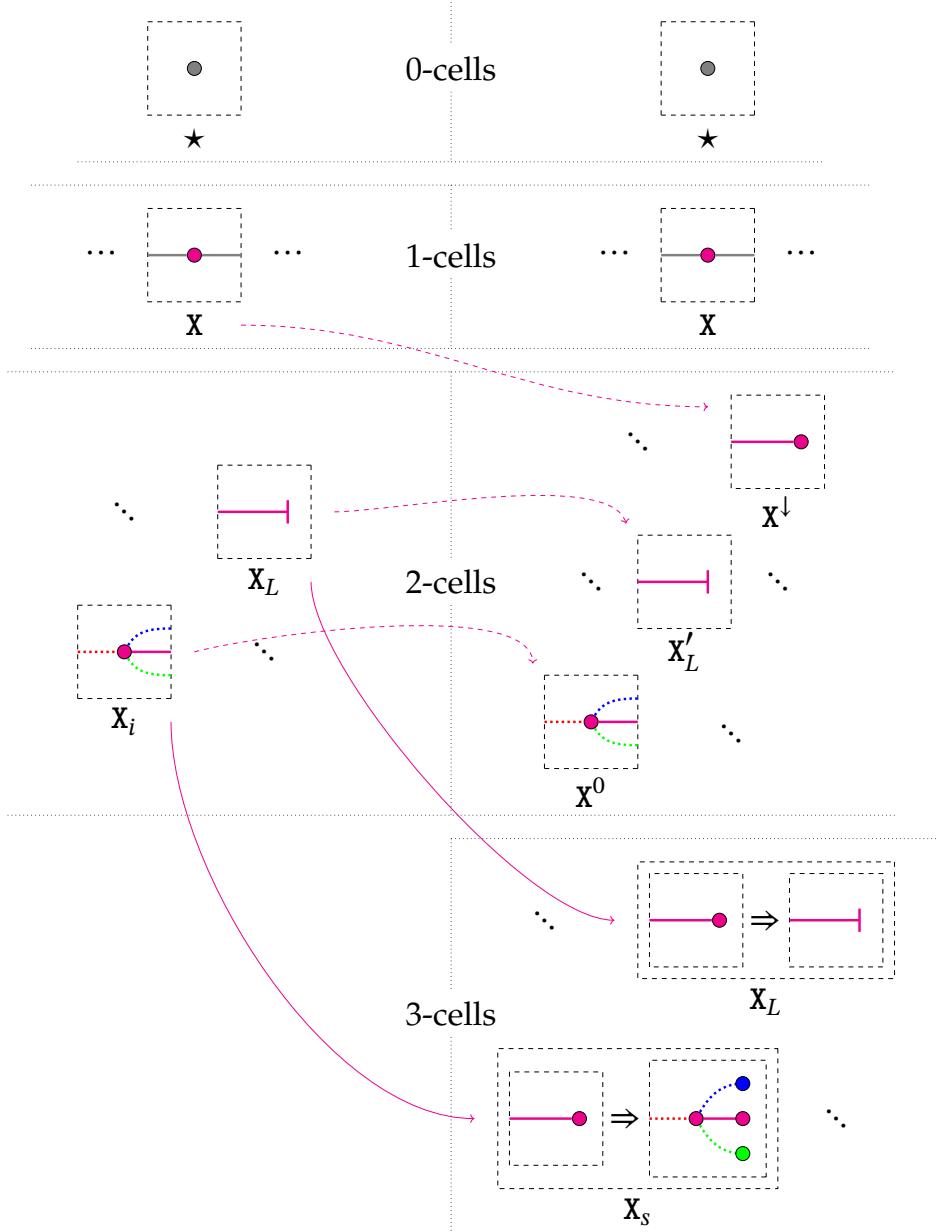
Construction 3.1.2 (Leaf-Ansatz of a CFG). Given a signature \mathfrak{G} for a CFG, we construct a new signature \mathfrak{G}' which has the same 0- and 1-cells as \mathfrak{G} . Now, referring to the dashed magenta arrows in the schematic below: for each 1-cell wire type X of \mathfrak{G} , we introduce a *leaf-ansatz* 2-cell X^\downarrow . For each leaf 2-cell X_L in \mathfrak{G} , we introduce a renamed copy X'_L in \mathfrak{G}' . Now refer to the solid magenta: we construct a 3-cell in \mathfrak{G}' for each 2-cell in \mathfrak{G} , which has the effect of systematically replacing open output wires in \mathfrak{G} with leaf-ansatzes in \mathfrak{G}' .

Proposition 3.1.3. Leaf-ansatzes of CFGs are precisely tree adjoining grammars (TAGs) with only initial trees and substitution.

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

Context-Free Grammar

Leaf-Ansatz



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

Figure 3.7: Instead of treating non-terminals as wires and terminals as effects (so that the presence of an open wire available for composition visually indicates non-terminality) the leaf-ansatz construction treats all symbols in a rewrite system as leaves, and the signature bookkeeps the distinction between nonterminals and terminals.

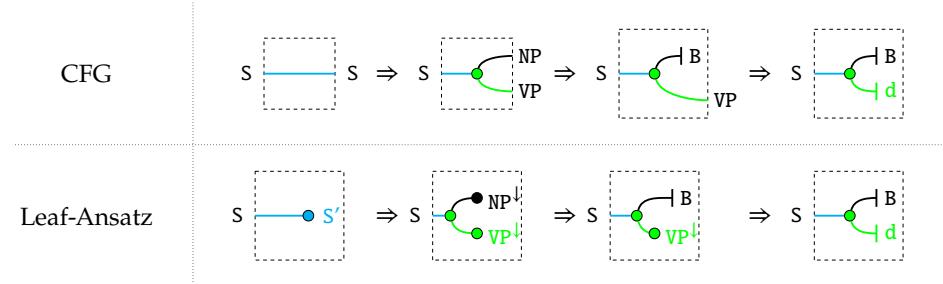


Figure 3.8: Adjoining is sprouting subtrees in the middle of branches. One way we might obtain the sentence Bob runs to school is to start from the simpler sentence Bob runs, and then refine the verb runs into runs to school. This refinement on part of an already completed sentence is not permitted in CFGs, since terminals can no longer be modified. The adjoining operation of TAGs gets around this constraint by permitting rewrites in the middle of trees.

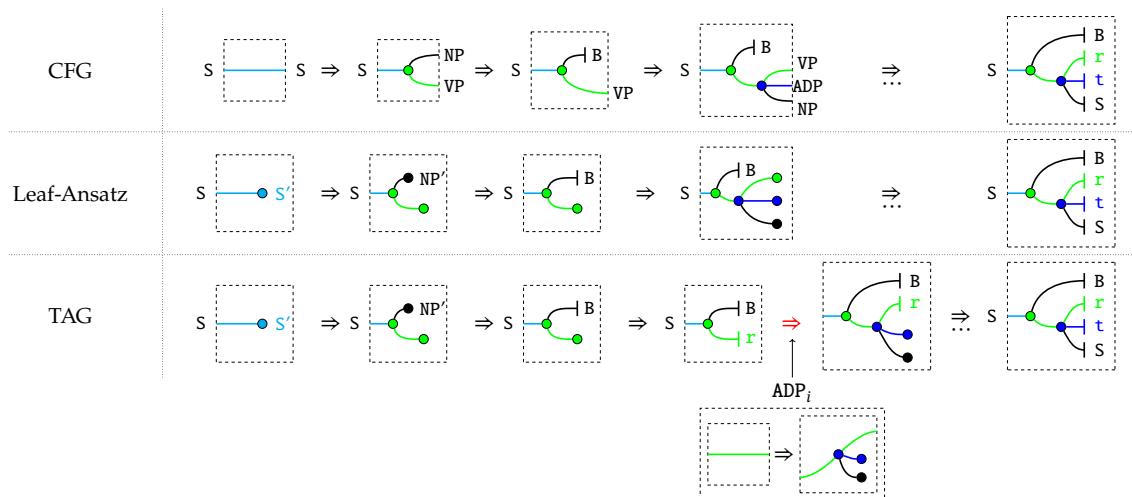


Figure 3.9: Leaf-ansatz signature of Alice sees Bob quickly run to school CFG.

1-cells correspond to types. There are three kinds of 2-cells organised in rows; terminals, substitutable ansätze that convert wires into bulbs; and the symbolic rewrites of CFGs respectively. There are two kinds of 3-cells similarly organised in rows; terminal-rewrites that replace a bulb with a terminal, and rewrites that mimic the CFG rewrites on ansätze. Note the one-to-one correspondence between the 2-cells and 3-cells for CFG rewrites and terminals.

In more detail, one aspect of rewrite systems we adapt for now is the distinction between terminal and nonterminal symbols; terminal symbols are those after which no further rewrites are possible. We capture this string-diagrammatically by modelling terminal rewrites as 2-cells with target equal to the 1-cell identity of the 0-cell \star , which amounts to graphically terminating a wire. The generators subscripted L (for *label* or *leaf*) correspond to terminals of the CFG, and represent a family of generators indexed by a lexicon for the language. The generators subscripted i (for introducing a type) correspond to rewrites of the CFG.

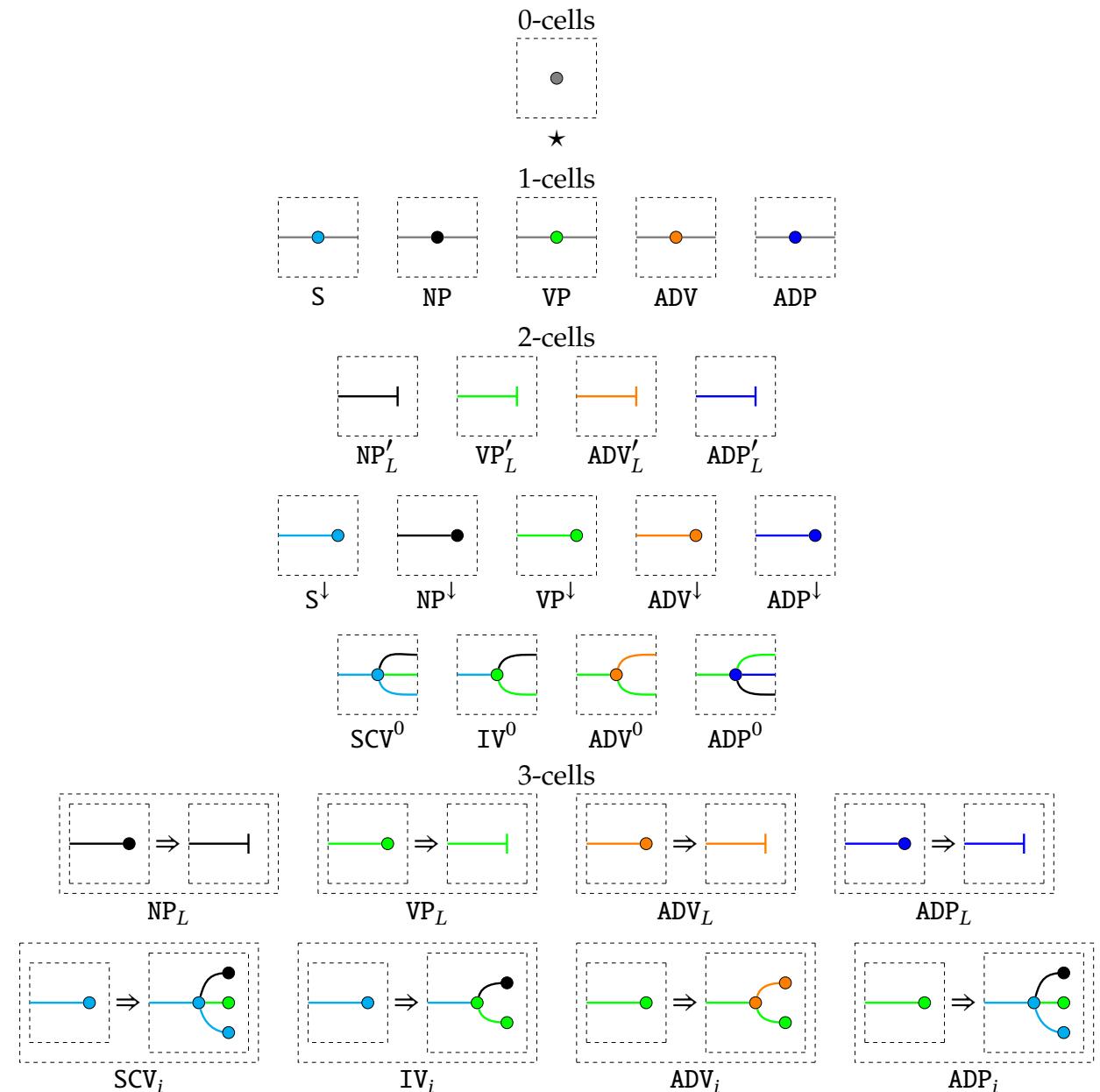
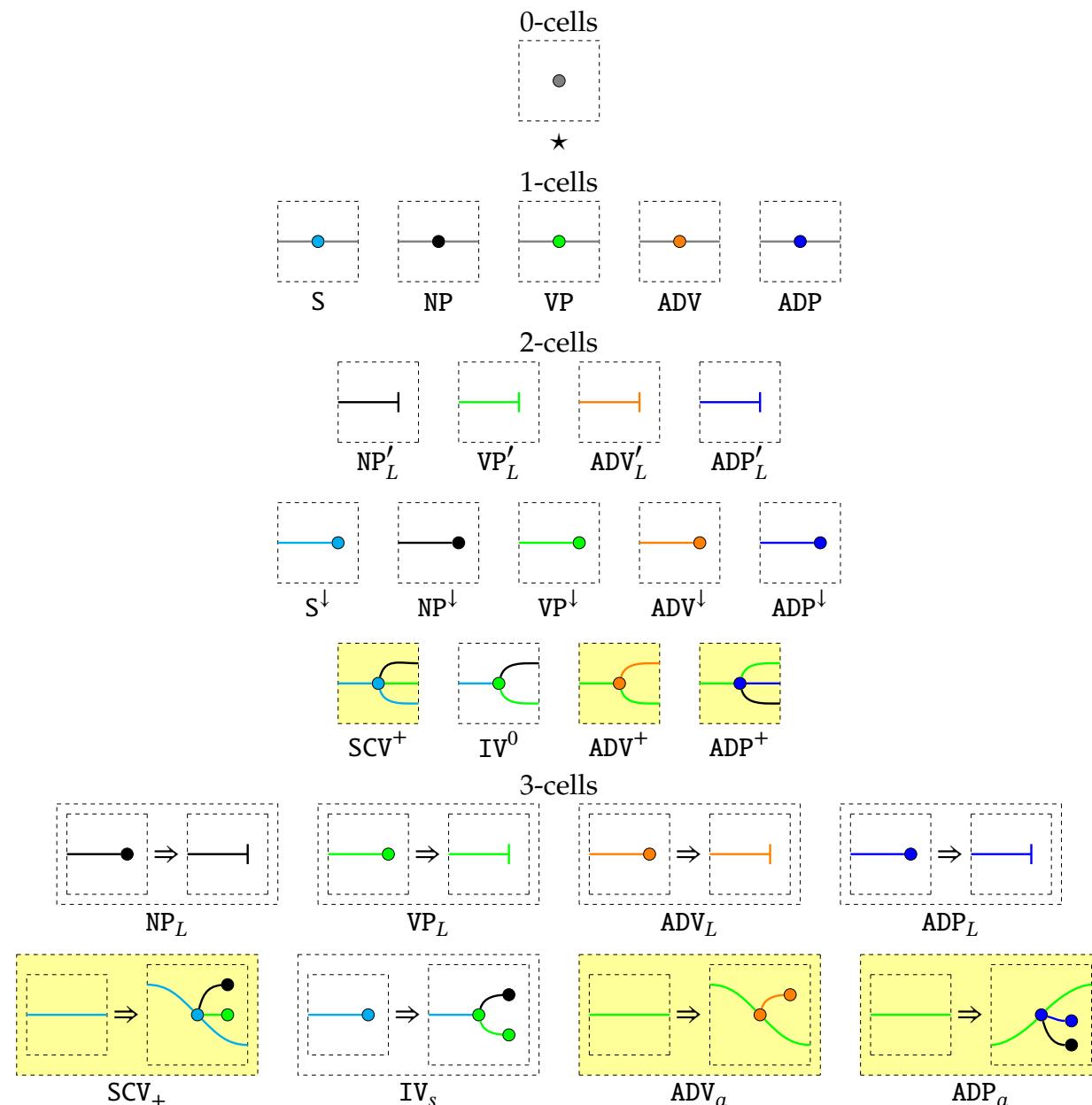


Figure 3.10: TAG signature of Alice sees Bob quickly run to school. The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees. The construction yields as a corollary an alternate proof of Theorem 6.1.1 of [Jos87] that recovers CFGs as TAGs.

Corollary 3.1.4. For every context-free grammar \mathcal{G} there exists a tree-adjoining grammar \mathcal{G}' such that \mathcal{G} and \mathcal{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 3.1.3 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 \mathcal{G}' corresponds to a single 2-cell tree of some CFG signature \mathcal{G} , which we demonstrate by construction. The highlighted 3-cells of \mathcal{G}' are obtained systematically from the auxiliary 2-cells as follows: the root and foot nodes X, X^* 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 Y^\downarrow . This establishes a correspondence between any 2-cells of \mathcal{G} considered as auxiliary trees in \mathcal{G}' . \square



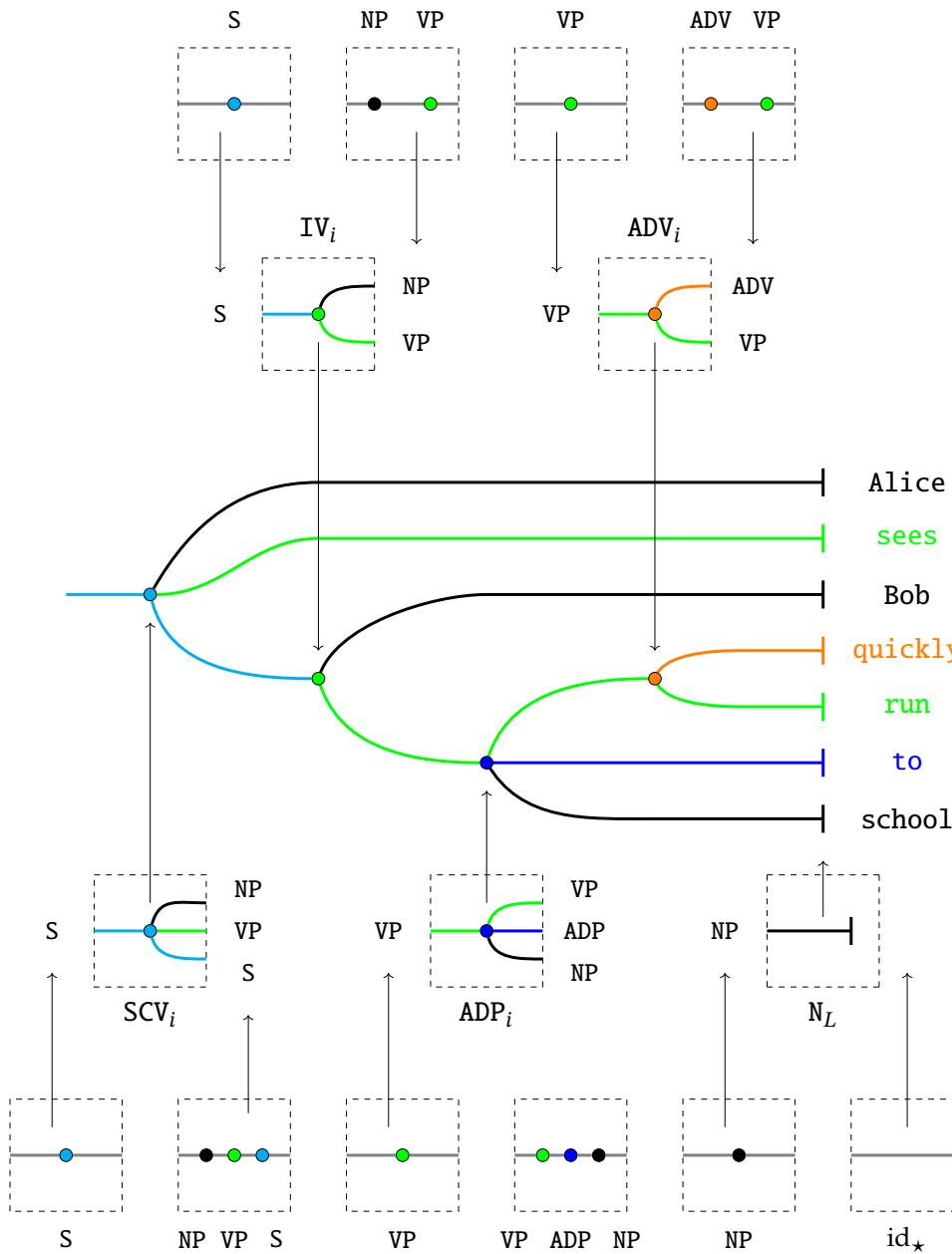


Figure 3.11: Reading the central diagram in the main body from left-to-right, we additionally depict the breakdown of the complete derivation in terms of the constituent 2-cells, and the source and target 1-cells. Evidently, all context-sensitive grammars may be viewed as finitely presented 1-object-2-categories by considering multi-input-multi-output rewrites. More broadly, any string rewriting system is recoverable in the presence of higher dimensional cells. My source for that is that I made a Turing machine in homotopy.io and executed busy-beaver on it as a homework exercise when Jamie Vicary taught Categorical Quantum Mechanics at Oxford.

Definition 3.1.5 (TAG with local constraints: CS-style). [Joshi] $G = (I, A)$ is a TAG with local constraints if for each node n and each tree t , exactly one of the following constraints is specified:

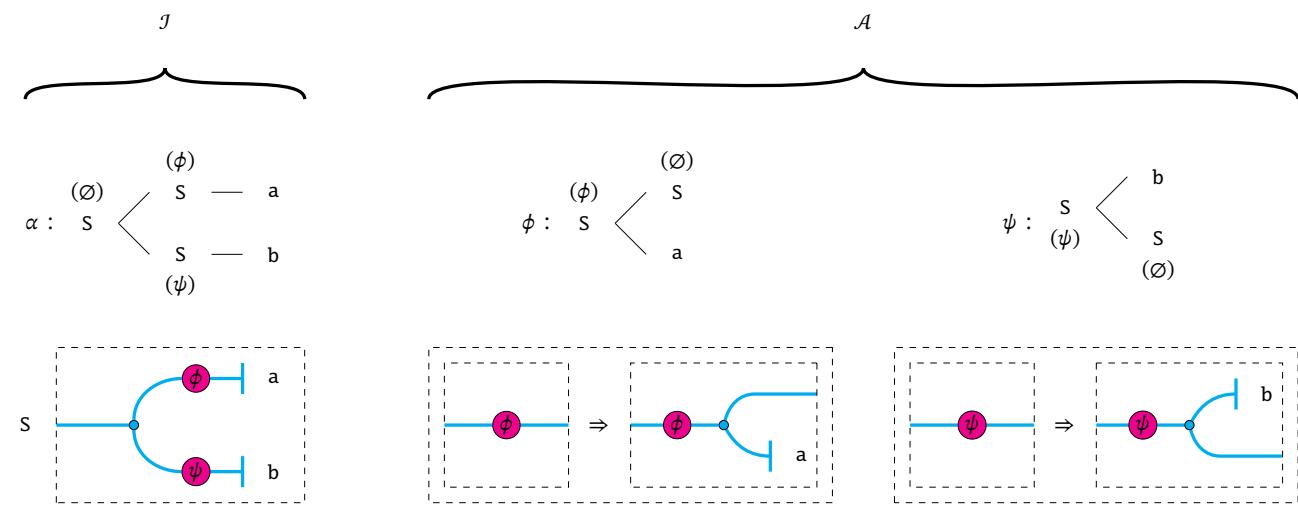
1. Selective adjoining (SA): Only a specified subset $\beta \subseteq A$ of all auxiliary trees are adjoinable at n .
2. Null adjoining (NA): No auxiliary tree is adjoinable at the node n .
3. Obligatory Adjoining (OA): At least one out of all the auxiliary trees adjoinable at n must be adjoined at n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

Figure 3.12: Selective and null adjoining diagrammatically: a reproduction of Example 2.5 of [Joshi] which demonstrates the usage of selective and null adjoining. The notation from [Joshi] is presented first, followed by their corresponding representations in an n -categorical signature. The initial tree is presented as a 2-cell where the (SA) rules are rewritable nodes, that serve as sources of rewrites in the 3-cell presentations of the auxiliary trees.

3.1.3 Tree adjoining grammars with local constraints

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

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

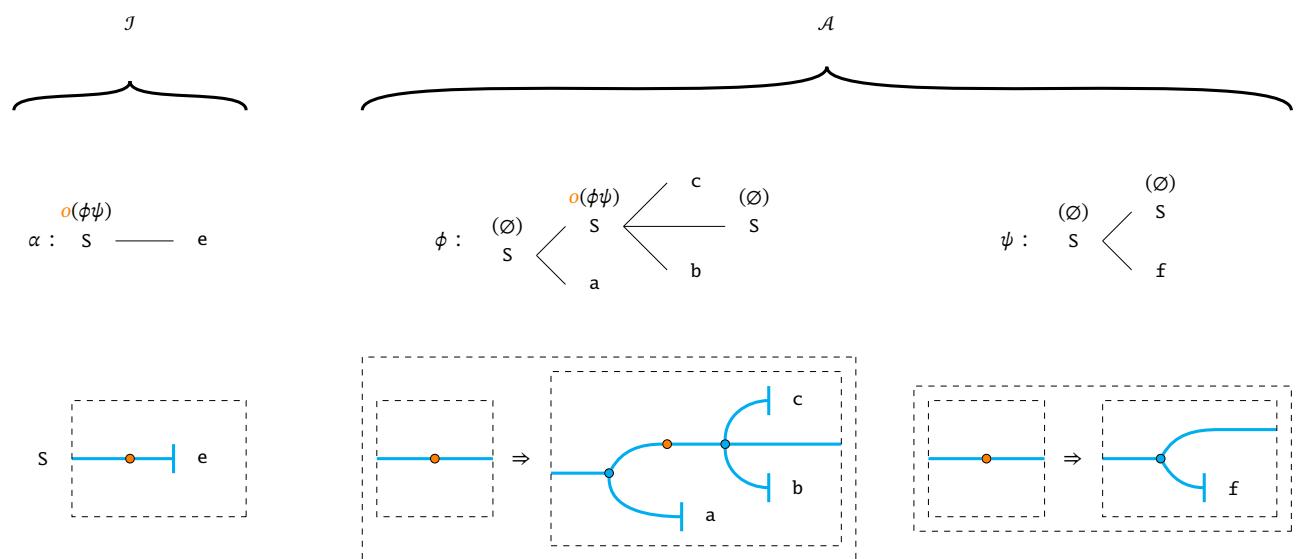


3.1.4 Braiding, symmetries, and suspension

Before we can model TAGs with links, we must introduce the concepts of braiding and symmetries, which we have seen in the diagrammatic setting already as wires twisting past one another. In our current setting of 1-object-3-categories for TAGs, the diagrams we are dealing with are all planar – i.e. wires may not cross – but this is a restriction we must overcome when we are dealing with links.

First we observe that in a 1-object-2-categorical setting, a morphism from the identity ϵ on the base object \star to itself would, in our analogy with string-rewrite systems, be a rewrite from the empty string to itself.

Figure 3.13: Obligatory adjoining diagrammatically: a reproduction of Example 2.11 of [Joshi] which demonstrates the usage of obligatory adjoining, marked orange. The notation from [Joshi] is presented first, followed by their corresponding representations in an n -categorical signature. The initial tree is presented as a 2-cell where the (OA) rule is given its own 2-cell, which is the source of rewrites in 3-cell presentations of auxiliary trees. We may capture the obligatory nature of the rewrite by asking that finished derivations contain no instance of the orange 2-cell. Such global acceptance conditions are hacky but common, and in this case it is efficiently verifiable that diagrams do not contain certain generators.



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

$$R := \varepsilon \mapsto x \mapsto \varepsilon$$

$$S := \varepsilon \mapsto a \cdot b \mapsto \varepsilon$$

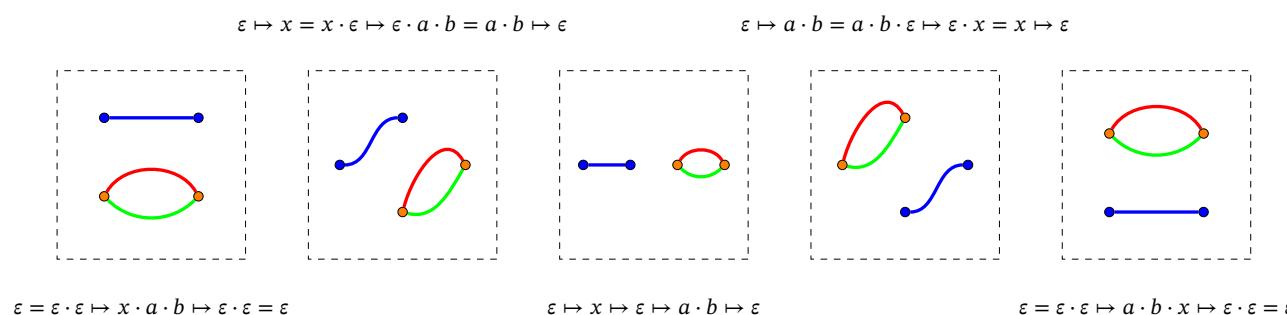


Figure 3.14: In our analogy with string rewrite systems, we might like that the following rewrites are equivalent, while respecting that they are not equal, representing x, a, b as blue, red, and green wires respectively. Such rewrites from the empty string to itself are more generally called *scalars* in the monoidal setting, viewed 2-categorically.

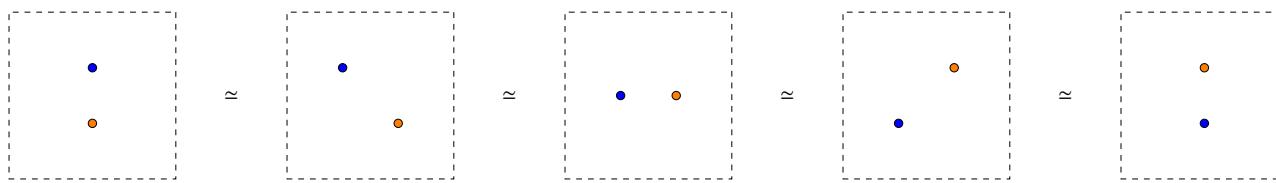


Figure 3.16: We may view the homotopies that get us from one rewrite to another as 3-cells, which produces a braid in a pair of wires when viewed as a vignette. Up to processive isotopies, which are continuous bijective transformations that don't let wires double back on themselves, we can identify two different braidings that are not continuously deformable to one another in the 3-dimensional space of the vignette. We distinguish the braidings visually by letting wires either go over or under one another.

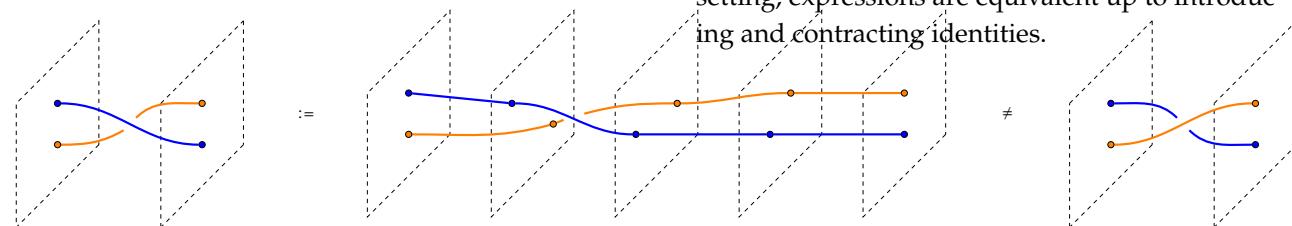


Figure 3.15: We may generally represent such scalars as labelled dots. A fact about scalars in a 1-object-2-category called the Eckmann-Hilton argument is that dots may circle around one another, and all of those expressions are equivalent up to homotopy. The mechanism that enables this in our setting 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.

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 1_{1_\star} . To obtain a dot in a 3-dimensional volume, we consider a rewrite from 1_{1_\star} to itself. These dots also enjoy a version of the Eckmann-Hilton argument in 3 dimensions (which holds for all dimensions 2 and higher).

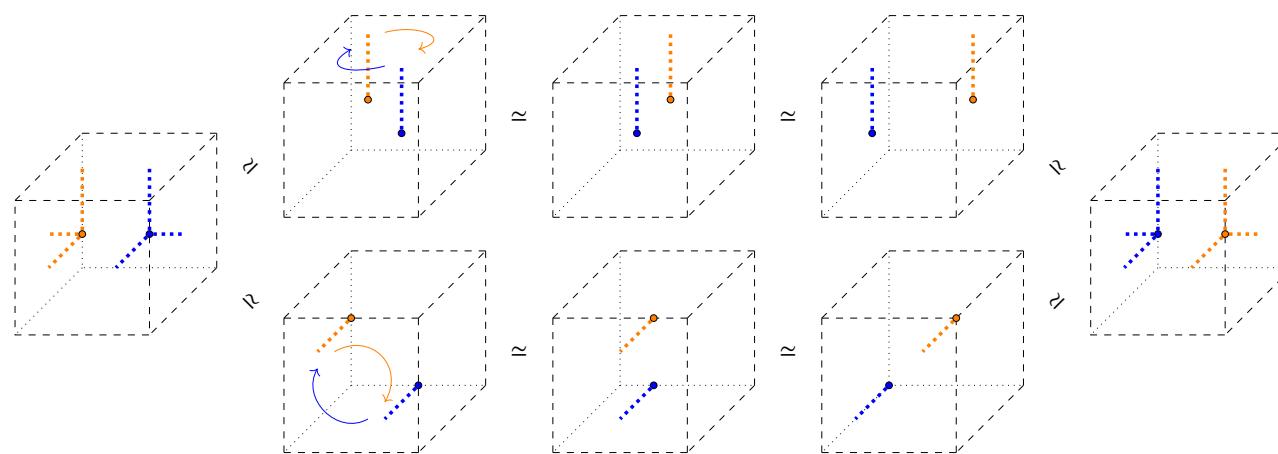


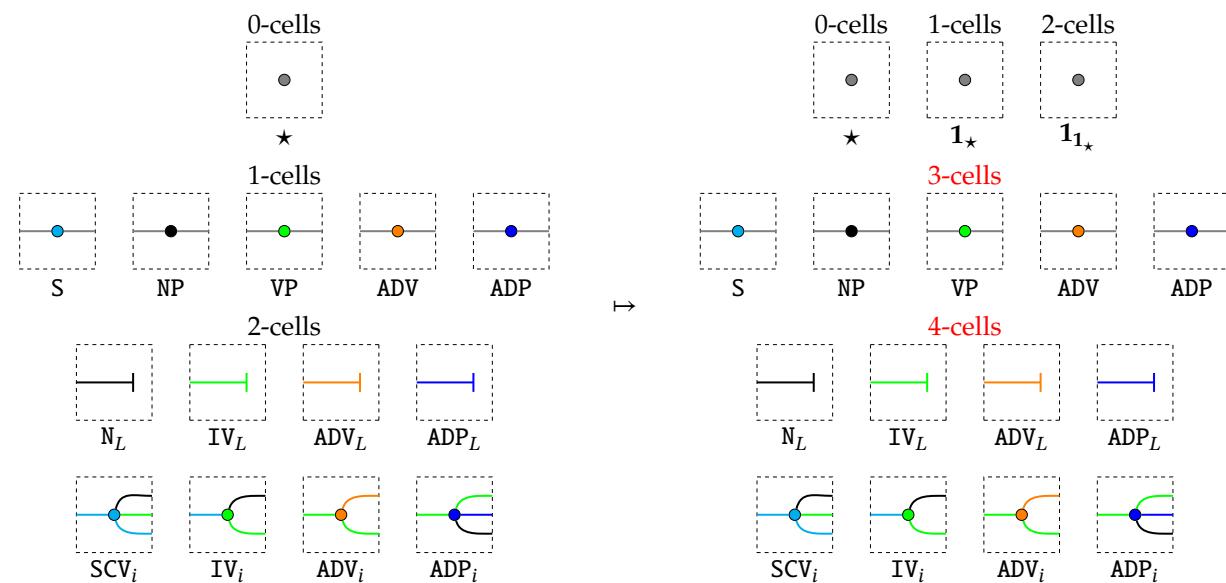
Figure 3.17: We can depict these swaps by movements in a cubic volume where each axis corresponds to a direction of composition. Whereas on the plane the dots have two ways to swap places – clockwise and counterclockwise rotation – in the volume they have two new ways to swap places – clockwise and counterclockwise in the new dimension. Shown below are two ways to swap left-to-right sequentially composed dots by clockwise rotations in the forward-backward and up-down directions of composition:

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

To recap: We follow the convention that object dimensions start at 0. In a 1-object-2-category we obtain planar string diagrams for monoidal categories, which are equivalent up to processive isotopies in an ambient 2-dimensional space. In a $(1 \times 0\text{-cell})-(1 \times 1\text{-cell})$ -3-category, we obtain string diagrams for braided monoidal categories that are equivalent up to processive isotopy in an ambient 3-dimensional space – the

page with depth. In a $(1 \times 0, 1 \times 1, 1 \times 2)$ -4 category, we obtain string diagrams for symmetric monoidal categories, which are equivalent up to processive isotopies in an ambient 4-dimensional space – while it is difficult to visualise 4-dimensionally, diagrammatically this simplifies dramatically to just allowing wires to cross and only caring about whether input-output connectivity from left to right makes sense. Now observe that for any 1-object-2-category, we can obtain a $(1 \times 0, 1 \times 1, 1 \times 2)$ -4 category by promoting all 2-cells and higher to sit on top of 1_{\star} .

Figure 3.18: For example, taking our CFG signature from earlier, suspension promotes 1-cells to 3-cells and 2-cells to 4-cells. The resulting signature gives us the same diagrams, now with the added ability to consider diagrams equivalent up to twisting wires, which models a string-rewrite system with free swapping of symbol order.



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

Remark 3.1.6 (THE IMPORTANT TAKEAWAY!). Now have a combinatoric way to specify string diagrams (that generalises PROPs, modulo weak interchange) for symmetric monoidal categories, which in addition permits us to reason with directed diagram rewrites. So we have a trinity of presentations of the same mathematical concept. String diagrams are the *syntax*. Symmetric monoidal categories are the *semantics*. n -categories are the *finite combinatoric presentation* of theories and rewrite systems upon them. By analogy, n -categories provide a *vocabulary* and *rewrites* for string-diagram *expressions*, which are interpreted with respect to the *context* of a symmetric monoidal category.

3.1.5 TAGs with links

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

Example 3.1.8. The following is a reproduction in `homotopy.io` of Example 2.4 of [Joshi], illustrating TAGs with links; as a proof assistant for weak n -categories, `homotopy.io` automatically performs tree adjunction after the signature has been properly spelled out.

Definition 3.1.7 (TAGs with links: CS-style). The following is a reproduction of the discussion on nodes with links from [Joshi]. Linking can be defined for any two nodes in an elementary tree. However, in the linguistic context we will require the following conditions to hold for a link in an elementary tree. If a node n_1 is linked to a node n_2 then:

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

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

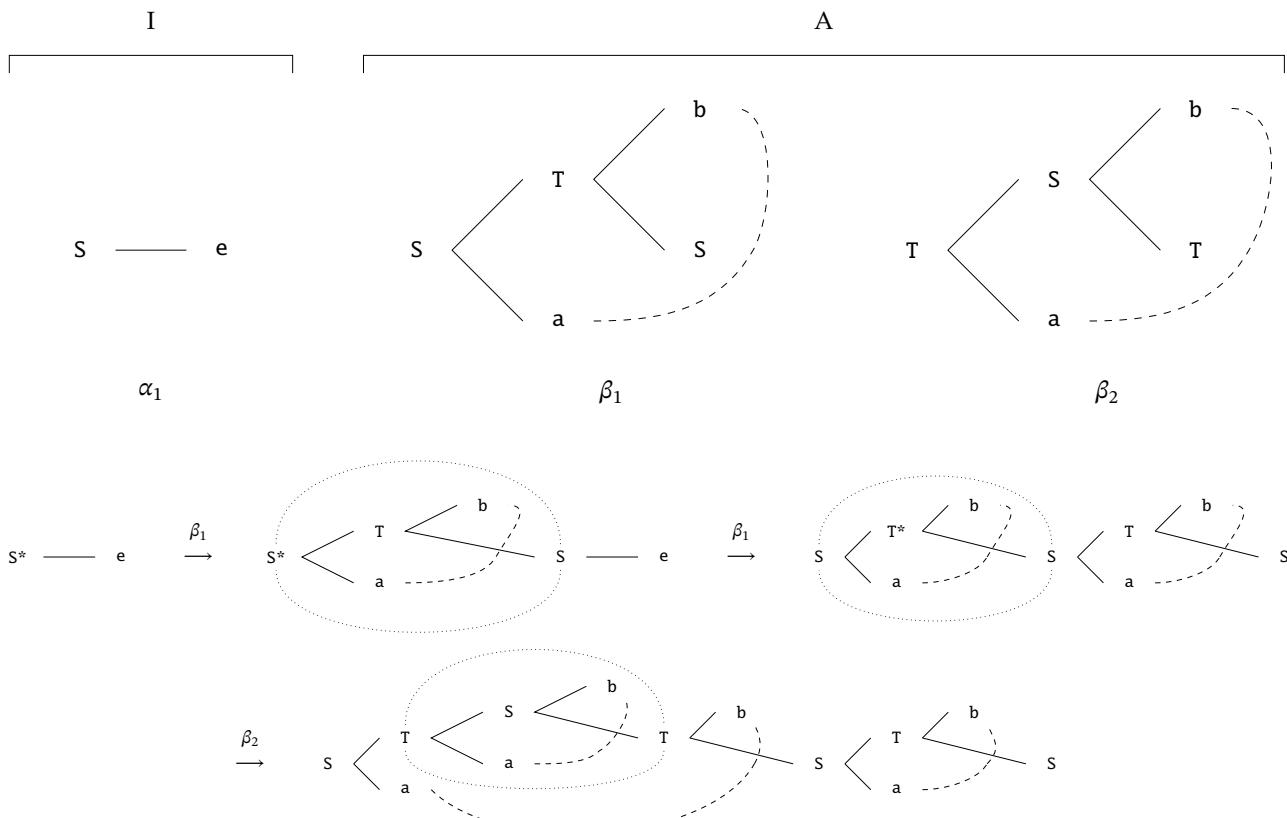


Figure 3.19: TAG signature and example derivation. Joshi stresses that adjoining *preserves* links, and that elementary trees may become *stretched* in the process of derivation, which are fundamentally topological constraints, akin to the "only (processive) connectivity matters" criterion identifying string diagrams up to isomorphism. Moreover, TAGs evidently have links of two natures: tree edges intended to be planar, and dashed dependency edges intended to freely cross over tree edges. It is easy, but a hack, to ask for planar processive isomorphisms for tree edges and extraplanar behaviour for dependency edges: these are evidently two different kinds of structure glued together, rather than facets of some whole. Weak n -categories offer a unified mathematical framework that natively accommodates the desired topological constraints while also granting expressive control over wire-types of differing behaviours. One method to recover TAGs true to the original conception is to stay in a planar 1-object-2-category setting while explicitly including wire-crossing cells for dependency links. The alternative method we opt for in Section 3.2 is to work in a pure "only connectivity matters" setting, recovering the linear ordering of words by generating cells along a chosen wire.

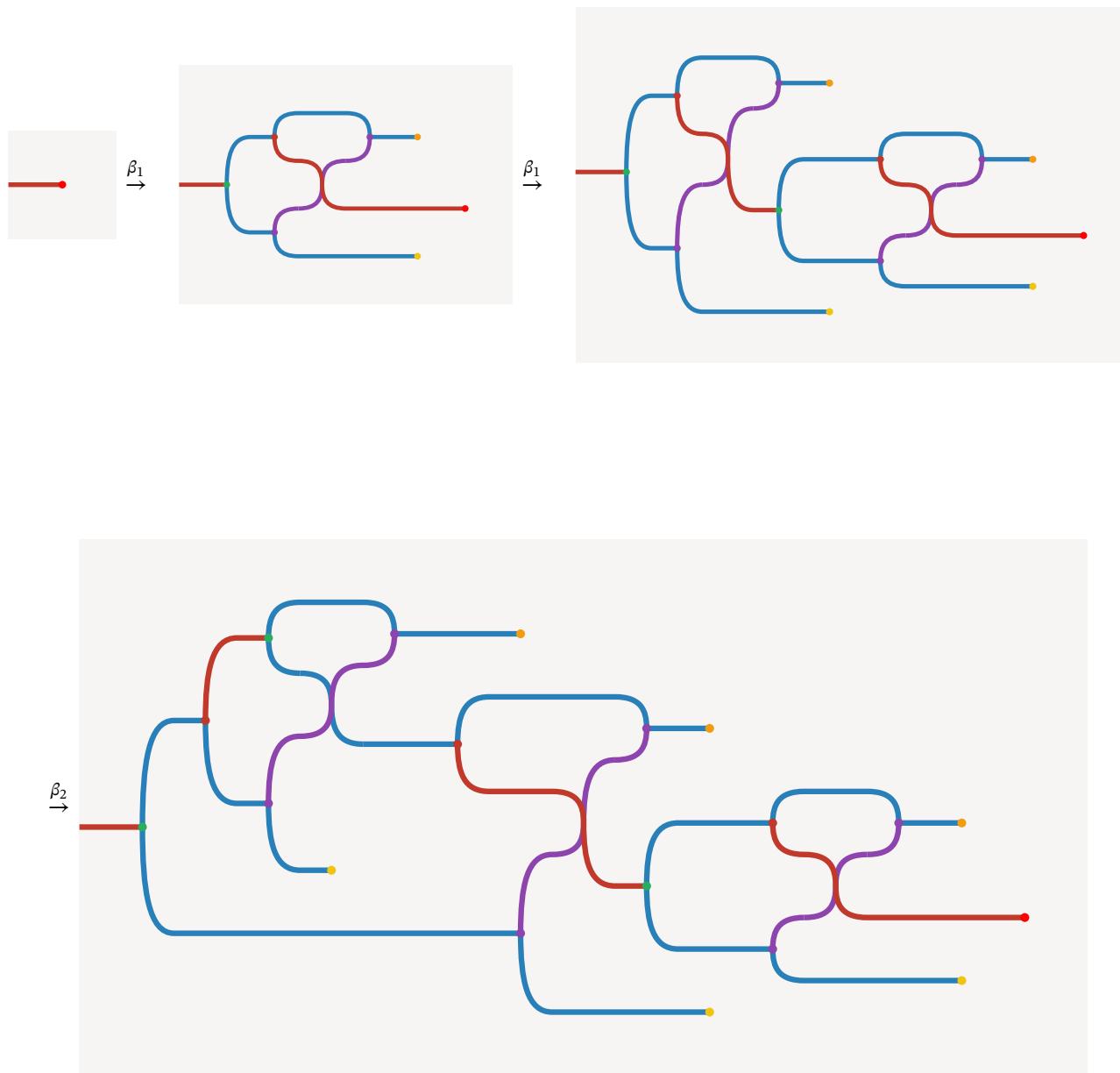


Figure 3.20: With our interpretation of TAGS as weak n -categorical signatures, We can recover each step of the same example derivation automagically in homotopy.io; just clicking on where we want rewrites allows the proof assistant to execute a typematching tree adjunction. In the process of interpretation, we introduce a link wire-type (in purple), and include directed link generation and elimination morphisms for the T wire-type (in blue). A necessary step in the process of interpretation (which for us involves taking a Poincaré dual to interpret nodes as wires) is a typing assignment of the tree-branches connected to terminal nodes, which we have opted to read as sharing a T -type for minimality, though we could just as well have introduced a separate label-type wire.

Figure 3.21: The intended takeaway is that even if you don't buy the necessity or formality of weak n -categories, there is always the fallback epistemic underpinning of a formal proof assistant for higher dimensional rewriting theories, which is rather simple to use if I have succeeded in communicating higher-dimensional intuitions in this section. **N.B.** In practice when using homotopy.io for the symmetric monoidal setting, it is simpler to suspend symmetric monoidal signatures to begin at 4-cells rather than 3-cells. The reason for this is that under- and over-braids still exist in the symmetric monoidal setting, and while sequentially composed braids are homotopically equivalent to the pair of identities, they are not uniquely so, thus these homotopies must be input manually. By beginning at 4-cells (or higher, due to the stabilisation hypothesis [nLad]), braid-eliminations are unique up to homotopy and can be performed more easily in the proof assistant.

Definition 3.1.9 (Linguistic Tree Adjoining Grammars). A *Linguistic Tree Adjoining Grammar* is given by the following data:

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

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

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

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 \emptyset), 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 $T \in \mathcal{N}$ and $T \in \mathfrak{L}$, or both n_1, n_2 are terminals.
 3. n_1 is the parent of terminal symbols, or childless.

3.1.6 Full TAGs in weak n -categories

I apologise in advance for the sheer ugliness of the following construction. Partly this is inevitable for combinatorial data, but the diagrammatic signatures also present combinatorial data and are easier to read, so what gives? This is what happens when one defines mathematical objects outside of their natural habitat. The native habitat of TAGs is diagrammatic, to accommodate a natural and spatially-intuitive generalisation of context-free grammars by allowing trees to be adjoined on nodes other than the leaves. I consider the offensiveness of the translation to follow a direct measure of the wrongness of linear-symbolic foundations, and an example of the harm that results from the prejudice that pictures cannot be formal.

Construction 3.1.10 (TAGs in homotopy.io). We spell out how the data of a TAG becomes an n -categorical signature by enumerating cell dimensions:

0. A single object \star
1. None.
2. None.
3. • For each $T \in \mathcal{N}$, a cell $T : \mathbf{1}_{1\star} \rightarrow \mathbf{1}_{1\star}$.
 - $T : \mathbf{1}_{1\star} \rightarrow \mathbf{1}_{1\star}$ A wire for terminal symbols.
 - For each $L \in \mathfrak{L}$, a cell $L : \mathbf{1}_{1\star} \rightarrow \mathbf{1}_{1\star}$.
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 $\sigma : T \rightarrow \mathbf{1}_{1_{1\star}}$.
 - Where $\phi \in \mathfrak{S}$ denotes a selective adjoining rule where required, if $n = (T, S, \dagger, *)$, we create $\phi S_T^\square : T \rightarrow T$ if the node is marked obligatory ($\dagger = \square$), and a cell $\phi S_T^\diamond : T \rightarrow T$ otherwise.
 - If $n = (T, S, \dagger, *)$, it is a foot node, for which we create a cell $S_T^\square : T \rightarrow \mathbf{1}_{1_{1\star}}$ if the node is marked obligatory ($\dagger = \square$), and a cell $S_T^\diamond : T \rightarrow \mathbf{1}_{1_{1\star}}$ otherwise.
 - For each $L \in \mathfrak{L}$ (which is also a type $T \in \mathcal{N}$) a pair of cells $T^L : T \rightarrow T \otimes L$ and $T_L : L \otimes T \rightarrow T$.

- For each node p of type T_p in either \mathcal{I} or \mathcal{A} with a nonempty left-to-right list of children $C_p := \langle c_1, c_2, \dots, c_i, c \dots c_n \rangle$ with types T_i , a branch cell $C_p :$
- $$T_p \rightarrow \bigotimes_{i=1}^n T_i.$$

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 $p = (T, S, \dagger)$, we begin with the cell S_T^\dagger . 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 T^L for the appropriate type T of the child node. Otherwise the child is c -commanded by a previously generated link, which we braid over and connect using T_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 $l = (T, S, \dagger)$ 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 T wire after appending S_T^\dagger . Altogether this obtains a 3-cell which we denote \mathcal{T} , overloading notation; different execution-orders of the above procedure evidently obtain 3-cells equivalent up to homotopy.

5. For each $\phi S_T^\square, \phi S_T^\diamond$, and for each typematching 4-cell tree in the subset of select adjoins of ϕ , we create a 5-cell rewrite that performs tree adjoining, taking the former to be the source and the latter to be the target.

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

3.2 A generative grammar for text circuits

3.2.1 A circuit-growing grammar

Definition 3.2.1 (Lexicon). We define a limited lexicon \mathcal{L} to be a tuple of disjoint finite sets $(\mathbf{N}, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_S, \mathbf{A}_N, \mathbf{A}_V, \mathbf{C})$

Where:

- \mathbf{N} is a set of *proper nouns*
- \mathbf{V}_1 is a set of *intransitive verbs*
- \mathbf{V}_2 is a set of *transitive verbs*
- \mathbf{V}_S is a set of *sentential-complement verbs*
- \mathbf{A}_N is a set of *adjectives*
- \mathbf{A}_V is a set of *adverbs*
- \mathbf{C} is a set of *conjunctions*

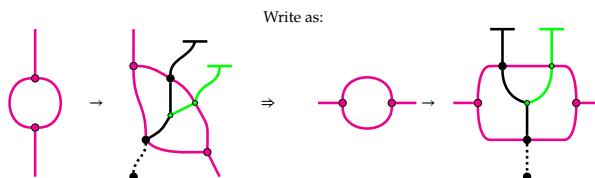


Figure 3.22: **How to read the diagrams in this section:** we will be making heavy use of pink and purple bubbles as frames to construct circuits. We will depict the bubbles horizontally, as we are permitted to by compact closure, or by reading diagrams with slightly skewed axes.

S-intro



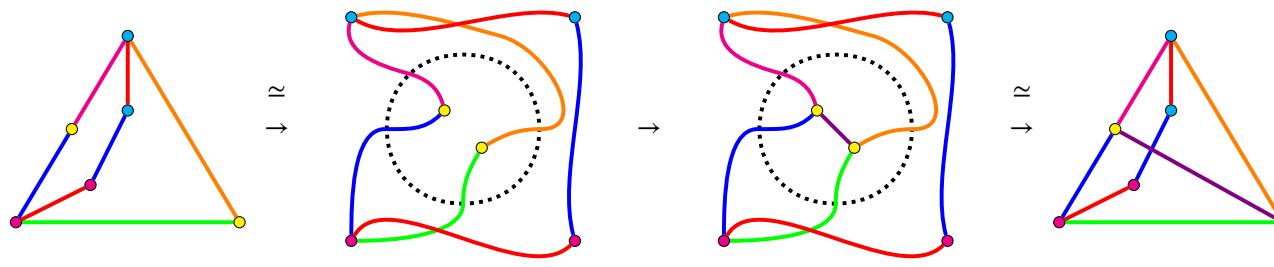
Figure 3.23: Every derivation starts with a single blank sentence bubble, to which we may append more blank sentences.

There are many different ways to write a weak n -categorical signature that generates circuits. Mostly as an illustration of expressivity, I will provide a signature where the terms "surface" and "deep" structure are taken literally as metaphors; the generative grammar will grow a line of words in syntactic order, and like mushrooms on soil, the circuits will behave as the mycelium underneath the words. It won't be the most efficient way to do it in terms of the number of rules to consider, but it will look nice and we'll be able to reason about it easily. As a note to the reader, there are a lot of worked examples in this section, so if you get confused about why a rule is the way it is, just skip over it and hopefully there will be a clarifying example later on.

SIMPLIFICATIONS AND LIMITATIONS: For now we only consider word types as in Definition 3.2.1, though we will see how to engineer extensions later. We only deal with propositional statements, without determiners, in only one tense, with no morphological agreement between nouns and their verbs and referring pronouns, and we assume that adverbs and adjectives stack indefinitely and without further order requirements: e.g. Alice happily secretly finds red big toy shiny car that he gives to Bob is a sentence we consider grammatical enough. For now, we consider only the case where adjectives and adverbs appear before their respective noun or verb. Note that all of these limitations apart from the limited lexicon can principle be overcome by the techniques we developed in Section 3.1 for restricted tree-adjoining and links. As a historical remark, generative-transformational grammars fell out of favour linguistically due to the problem of overgeneration: the generation of nonsense or unacceptable sentences in actual language use. We're undergenerating and overgenerating at the same time, but we're also not concerned with empirical capture: we only require a concrete mathematical basis to build interesting things on top of. On a related note, there's zero chance that this particular circuit-growing grammar even comes close to how language is actually produced by humans, and I have no idea whether a generalised graph-rewriting approach is cognitively realistic.

MATHEMATICAL ASSUMPTIONS: We work in a dimension where wires behave symmetric monoidally by homotopy, and further assume strong compact closure rewrite rules for all wire-types. Our strategy will be to generate "bubbles" for sentences, within which we can grow circuit structure piecemeal. We will only express the rewrite rules; the generators of lower dimension are implicit. We aim to recover the linear ordering of words in text (essential to any syntax) by traversing the top surface of a chain of bubbles representing sentence structure in text – this order will be invariant up to compact closed isomorphisms. The diagrammatic consequence of these assumptions is that we will be working with a conservative generalisation of graph-rewriting defined by local rewriting rules. The major distinction is that locality can be redefined up

to homotopy, which allows locally-defined rules to operate in what would be a nonlocal fashion in terms of graph neighbourhoods, as in Figure 3.24. The minor distinction is that rewrite rules are sensitive to twists in wires and the radial order in which wires emanate from nodes, though it is easy to see how these distinctions can be circumvented by additional by imposing the equivalent of commutativity relations as bidirectional rewrites. It is worth remarking that one can devise weak n-categorical signatures to simulate turing machines, where output strings are e.g. 0-cells on a selected 1-cell, so rewrite systems of the kind we propose here are almost certainly expressively sufficient for anything; the real benefit is the interpretable geometric intuitions of the diagrams.



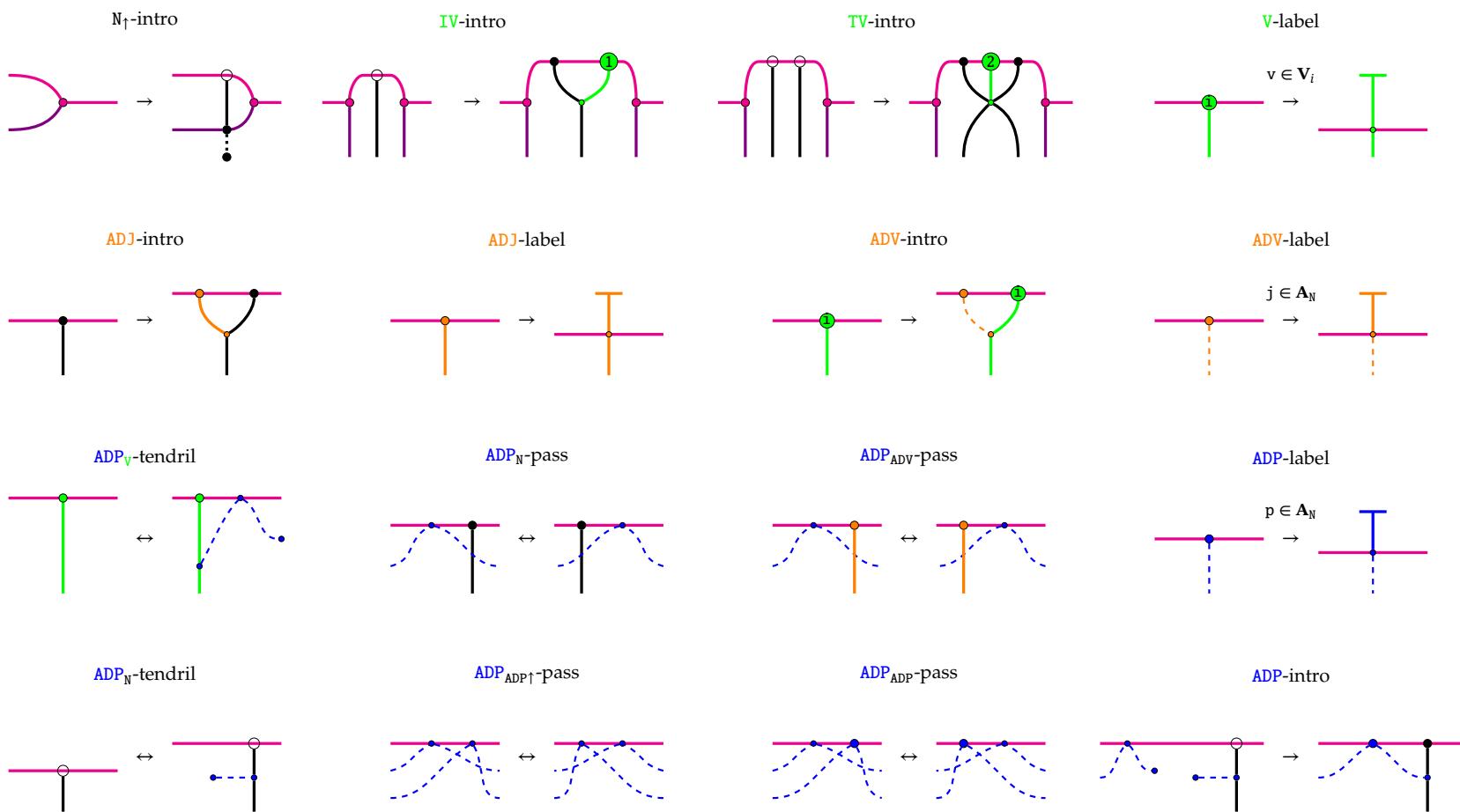
THE BROAD PLAN: We'll first display the rules and demonstrate their usage, then we'll prove the text circuit theorem by relating our rules to text.

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

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

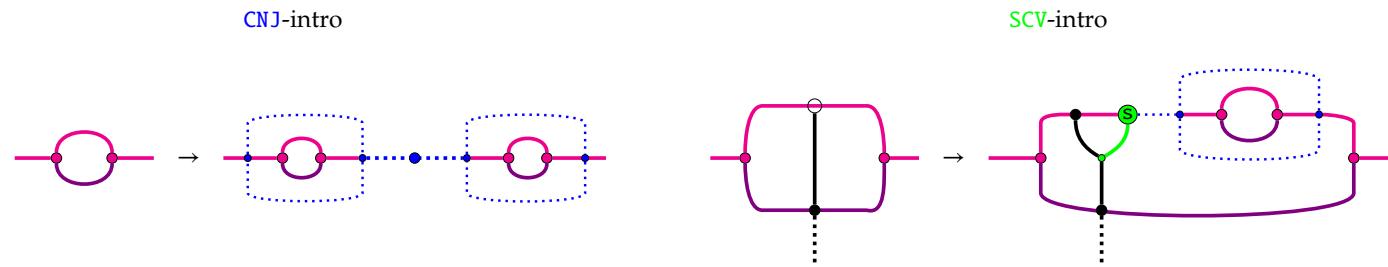
Figure 3.24: In this toy example, obtaining the same rewrite that connects the two yellow nodes with a purple wire using only graph-theoretically-local rewrites could potentially require an infinite family of rules for all possible configurations of pink and cyan nodes that separate the yellow, or would otherwise require disturbing other nodes in the rewrite process. In our setting, strong compact closure homotopies handle navigation between different spatial presentations so that a single rewrite rule suffices: the source and target notated by dotted-black circles. Despite the expressive economy and power of finitely presented signatures, we cannot "computationally cheat" graph isomorphism: formally we must supply the compact-closure homotopies as part of the rewrite, absorbed and hidden here by the \approx notation.

Rules 3.2.2 (Simple sentences). Simple sentences are sentences that only contain a single intransitive or transitive verb. Simple sentences will contain at least one noun, and may optionally contain adjectives, adverbs, and adpositions. The rules for generating simple sentences are as follows:



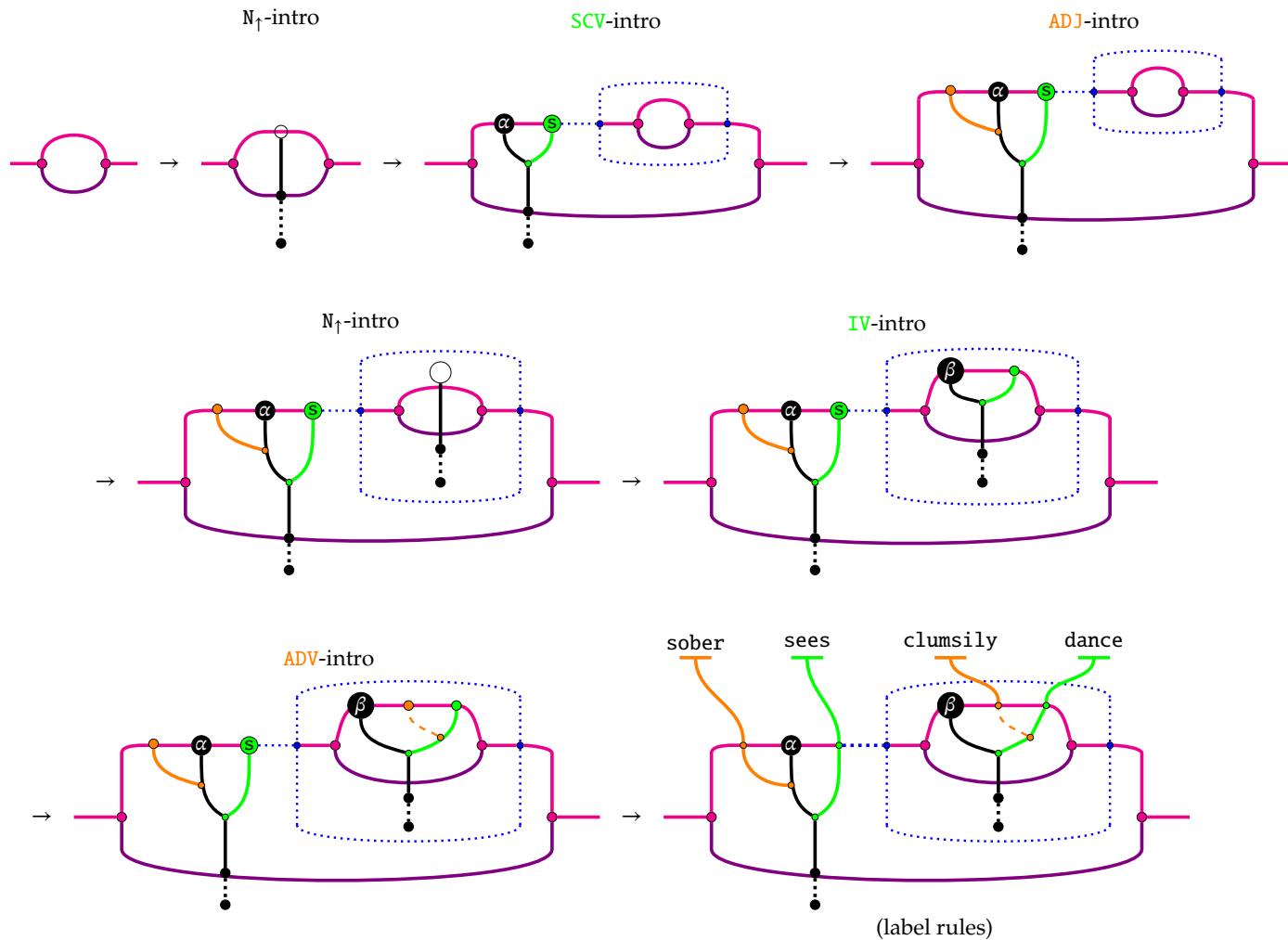
The **N_↑-intro** rule introduces new unsaturated nouns. The **IV-intro** and **TV-intro** rules apply when there are precisely one or two unsaturated nouns in the sentence respectively, saturating their respective nouns. Adjectives may be introduced immediately preceding saturated nouns. Adverbs may be introduced immediately preceding verbs. To capture context-sensitive placement of adposition introductions, the **ADP_v-tendril** rule allows an unsaturated adposition to succeed a verb; a bulb may travel by homotopy to the right seeking an unsaturated noun. Conversely, the bidirectional **ADP_n-tendril** rule sends a myclic tendril to the left, seeking a verb. The two pass-rules allow unsaturated adpositions to swap past saturated nouns and adjectives. By construction, neither verbs nor adverbs will appear in a simple sentence to the right of a verb, so unsaturated adpositions will move right until encountering an unsaturated noun. In case it doesn't, the tendril- and pass- rules are reversible.

Rules 3.2.3 (Complex sentences). Now we consider two refinements; conjunctions, and verbs that take sentential complements. We may have two sentences joined by a conjunction, e.g. Alice dances while Bob drinks. We may also have verbs that take a sentential complement rather than a noun phrase, e.g. Alice sees Bob dance; these verbs require nouns, which we depict as wires spanning bubbles.

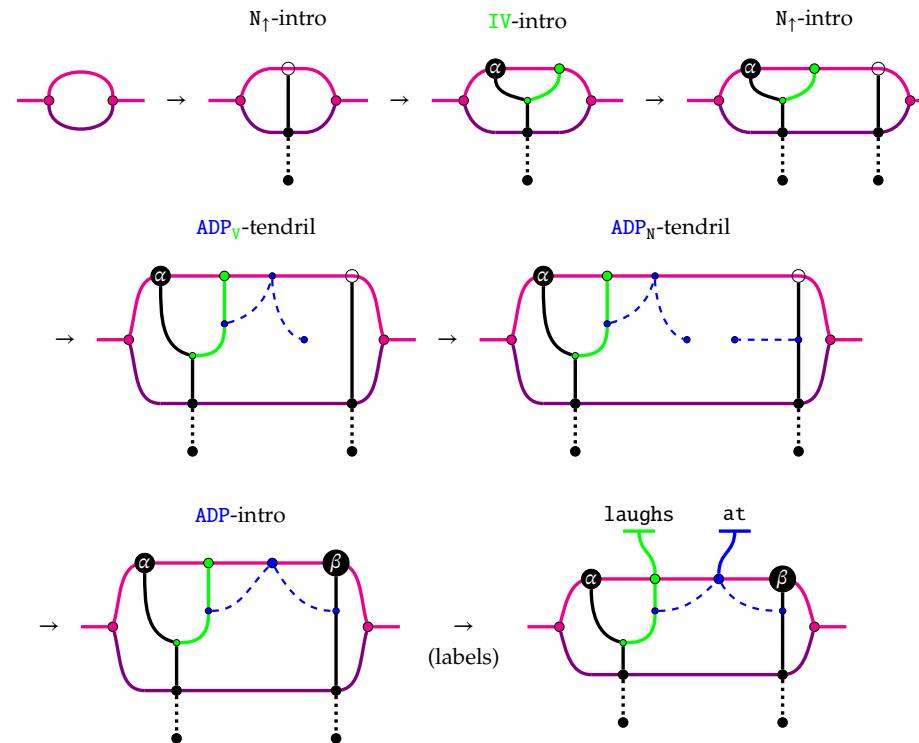


The dotted-blue wires do not contentfully interact with anything else, but the points at which they connect on the surface magenta wire serve as blockers that prevent overgeneration cases where adpositional phrases might interject between SCV verbs and their sentential complement, e.g. **Alice sees at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

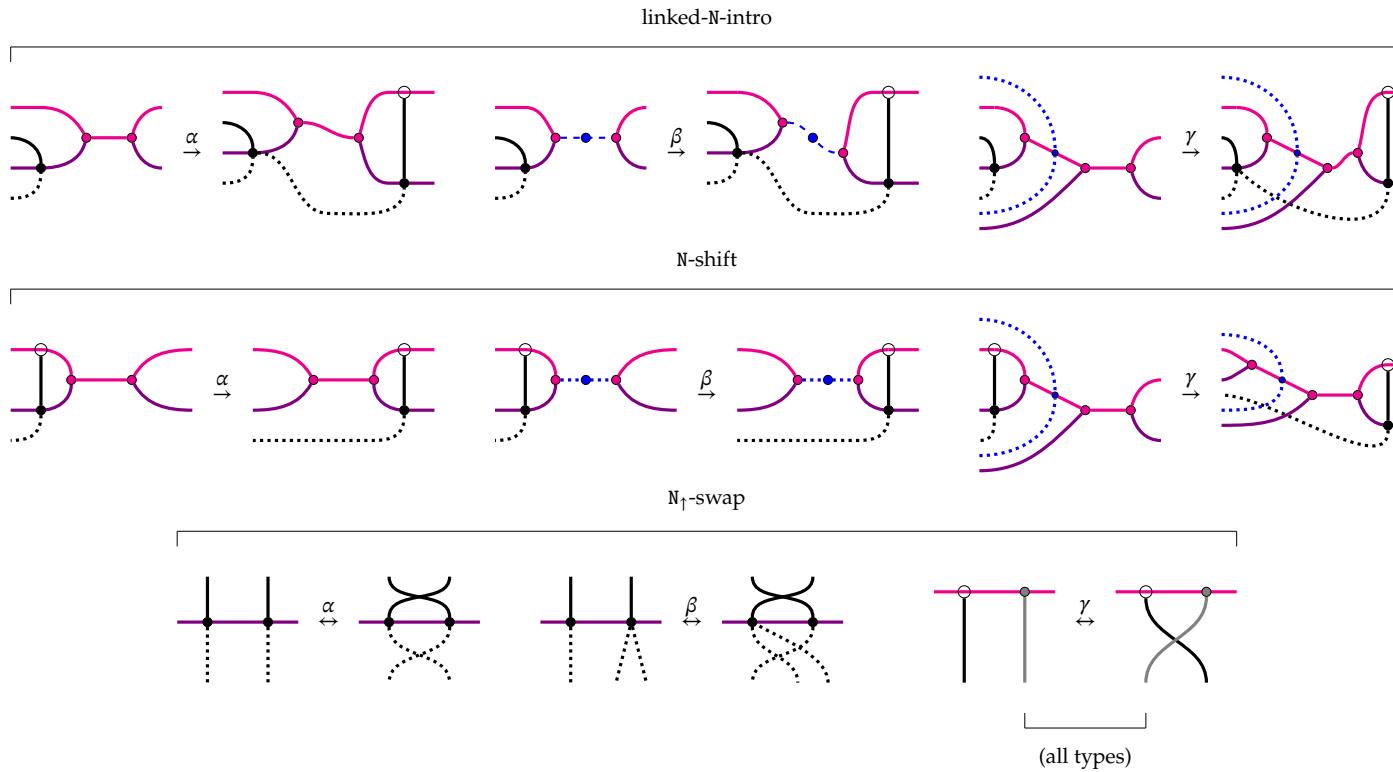
Example 3.2.4 (sober α sees drunk β clumsily dance.). Now we can see our rewrites in action for sentences. As a matter of convention – reflected in how the various pass- rules do not interact with labels – we assume that labelling occurs after all of the words are saturated. We have still not introduced rules for labelling nouns: we delay their consideration until we have settled coreferential structure. For now they are labelled informally with greeks.



Example 3.2.5 (α laughs at β). Adpositions form by first sprouting and connecting tendrils under the surface. Because the tendril- and pass- rules are bidirectional, extraneous tendrils can always be retracted, and failed attempts for verbs to find an adpositional unsaturated noun argument can be undone. Though this seems computationally wasteful, it is commonplace in generative grammars to have the grammar overgenerate and later define the set of sentences by restriction, which is reasonable so long as computing the restriction is not computationally hard. In our case, observe that once a verb has been introduced and its argument nouns have been saturated, only the introduction of adpositions can saturate additionally introduced unsaturated nouns. Therefore we may define the finished sentences of the circuit-growing grammar to be those that e.g. contain no unsaturated nodes on the surface, which is a very plausible linear-time check by traversing the surface. It is an invariant of the rules by construction that left-to-right traversal of the surface is always meaningful and yields the desired linear ordering of text in finished derivations.



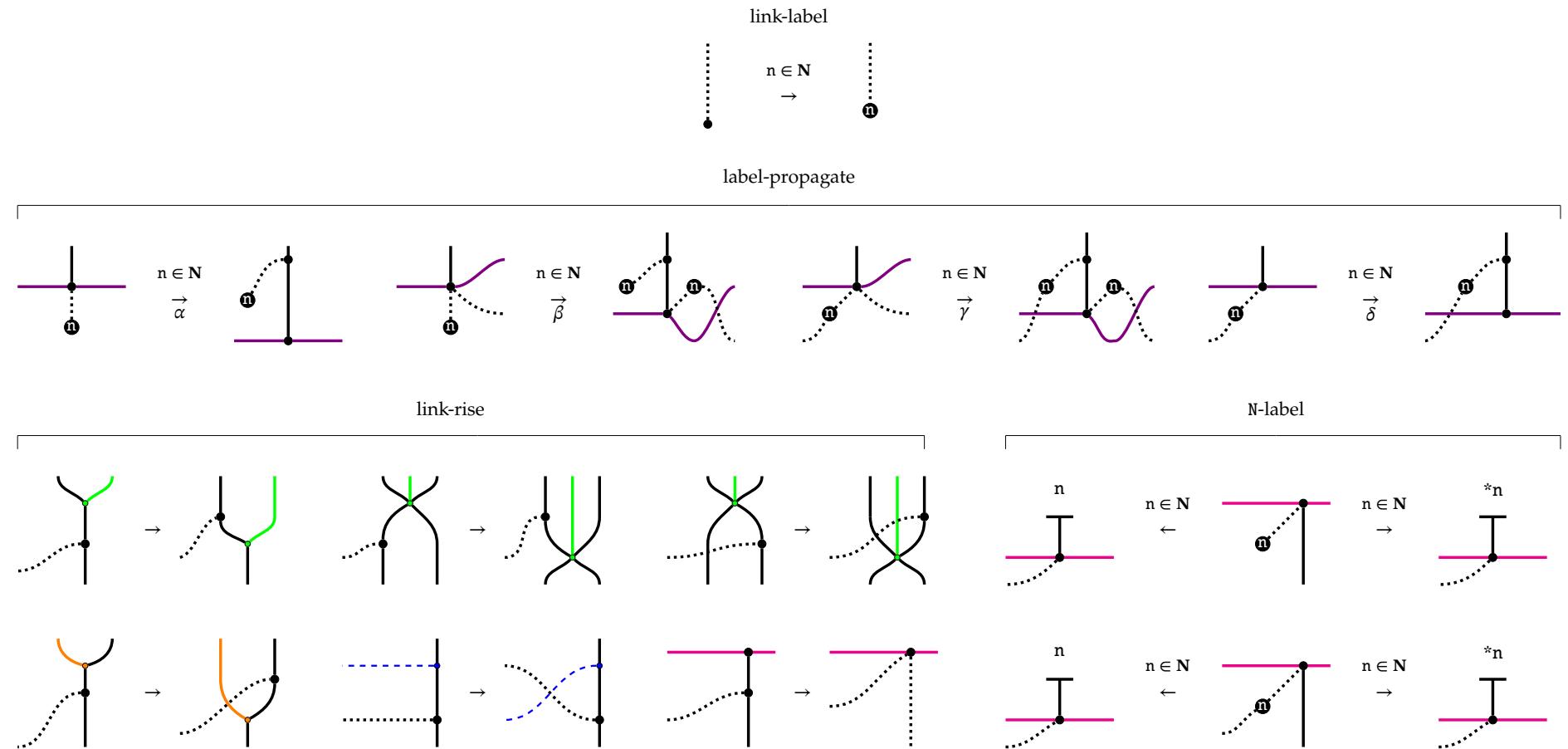
Rules 3.2.6 (Coreferential structure and noun labels).



The linked-N-intro rules introduce a new unsaturated noun in the next sentence that coreferences the noun in the previous sentence that generated it, with variants for each pair of sentences involved. We depict three: simple to simple, between CNJ-related sentences, and from either a CNJ or SCV to a simple sentence. N-shift rules allow any unsaturated noun to move into the next sentence, again with variants for different pairs of sentences. Observe that nouns with a forward coreference have two dotted-black wires leaving the root of their wires, which distinguishes them from nouns that only have a backward coreference or no coreference at all, which only have a single dotted-black wire leaving the root of their wire.

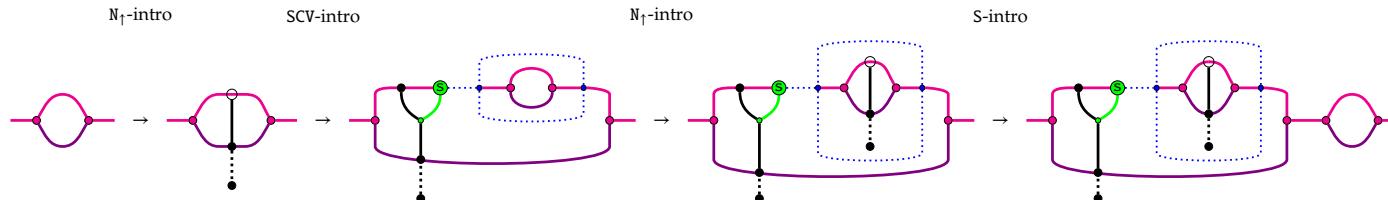
The N-swap rule variants allow a unsaturated noun with no forward coreferences to swap places with any unsaturated noun that immediately succeeds it.

Rules 3.2.7 (Labelling nouns). When the structure of coreferences is set, we propagate noun labels from the head of each list. The rules for noun-label propagation are as follows:

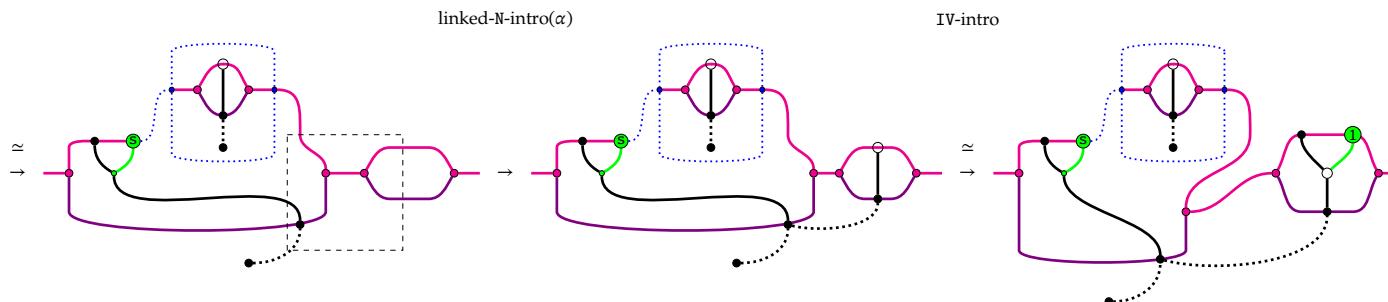


The $n \in N$ notation indicates a family of rewrites (and generators) for each noun in the lexicon. Link-label assigns a noun to a diagrammatically linked collection of coreferent nouns, and link-propagation is a case analysis that copies a link label and distributes it across coreferent nouns. Link-rise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

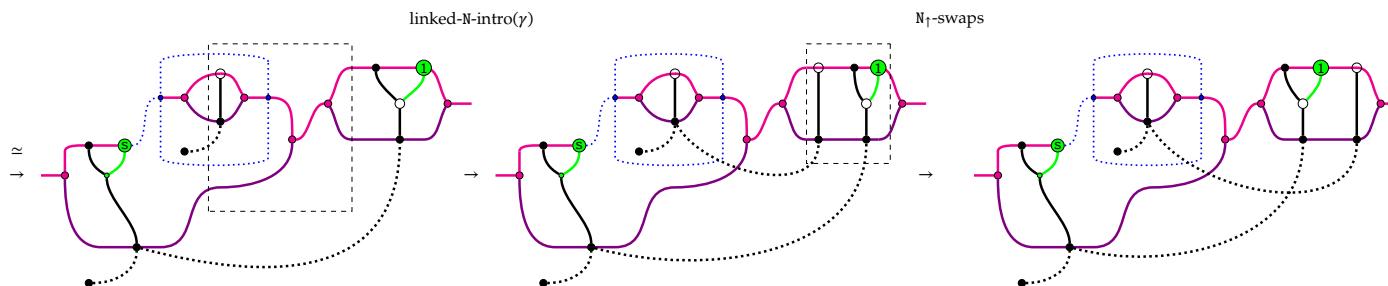
Example 3.2.8 (sober Alice sees Bob clumsily dance. She laughs at him.). We start the derivation by setting up the sentence structure using S- and SCV-intro rules, and two instances of N-intro, one for Alice, and one for Bob. Observe how the N-intro for Bob occurs within the subsentence scoped over by the SCV-rule.



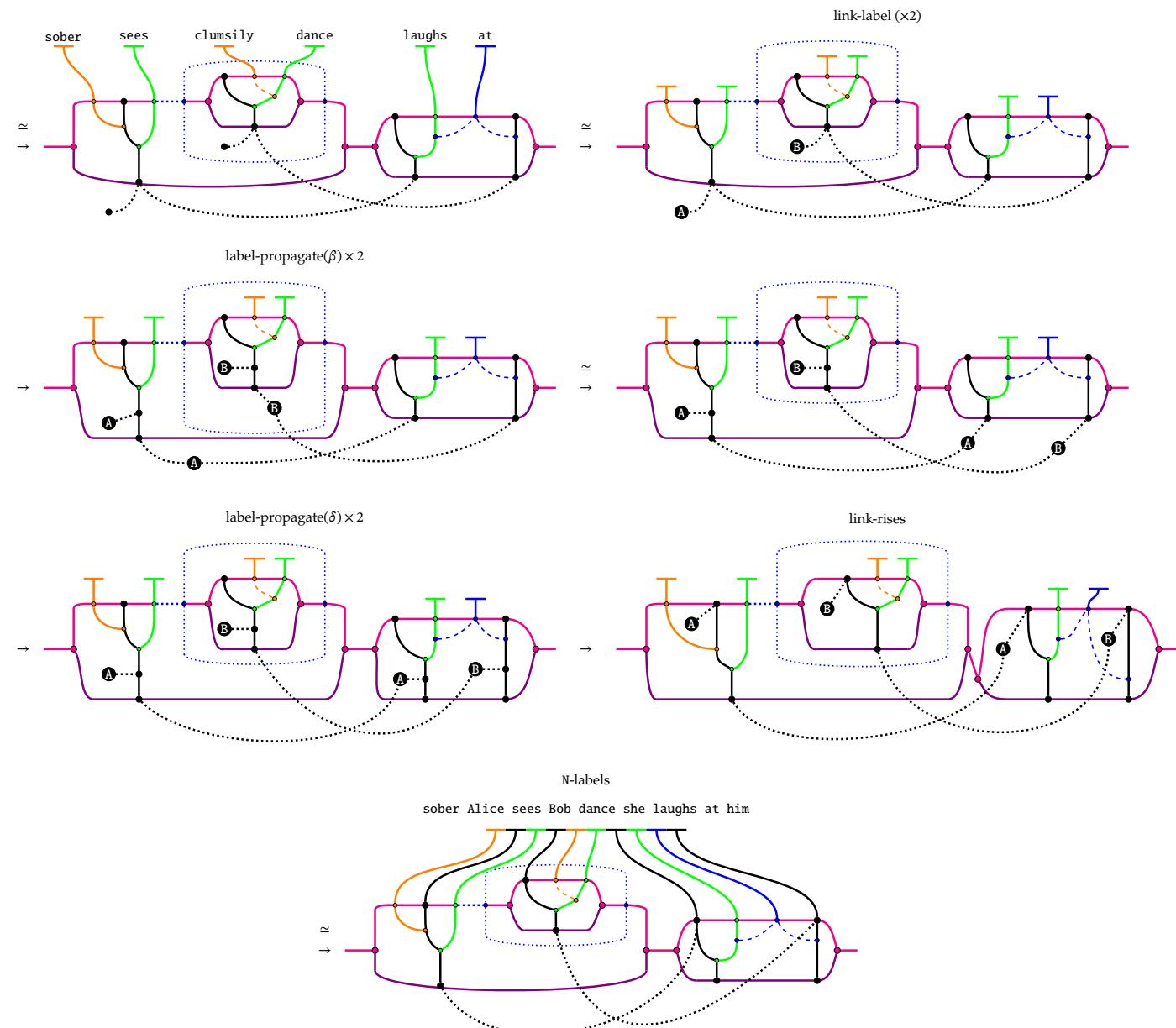
By homotopy, we can rearrange the previous diagram to obtain the source of the linked-N-intro rewrite in the dashed-box visual aid. Observe how we drag in the root of what is to be Alice's wire. Then we use the IV-intro in the second sentence, which sets up the surface structure she laughs, and the deep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



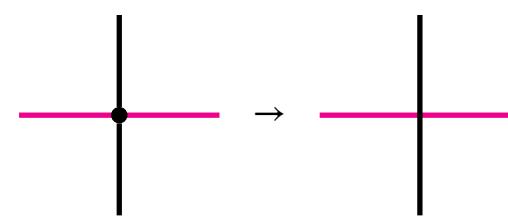
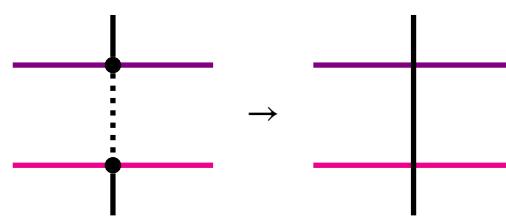
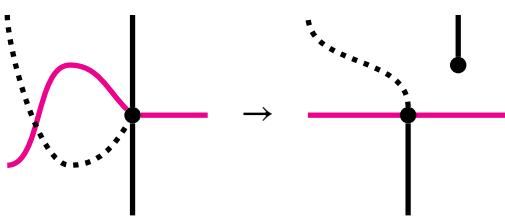
By homotopy again, we can do the same for Bob, this time setting up for the γ variant of linked-N-intro which handles the case when the spawning noun is within the scope of an SCV. Then by applying a series of N_1 -swaps, the unsaturated noun is placed to the right of the intransitive verb phrase.



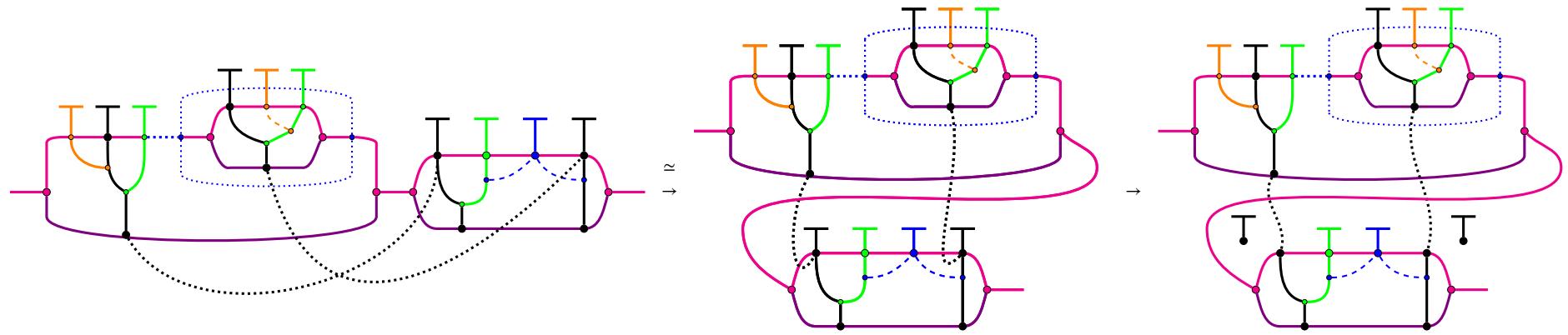
We've already done the surface derivation for the two sentences separately in Examples 3.2.4 and 3.2.5; since neither of those derivations touch the roots of noun-wires, we can emulate those derivations and skip ahead.



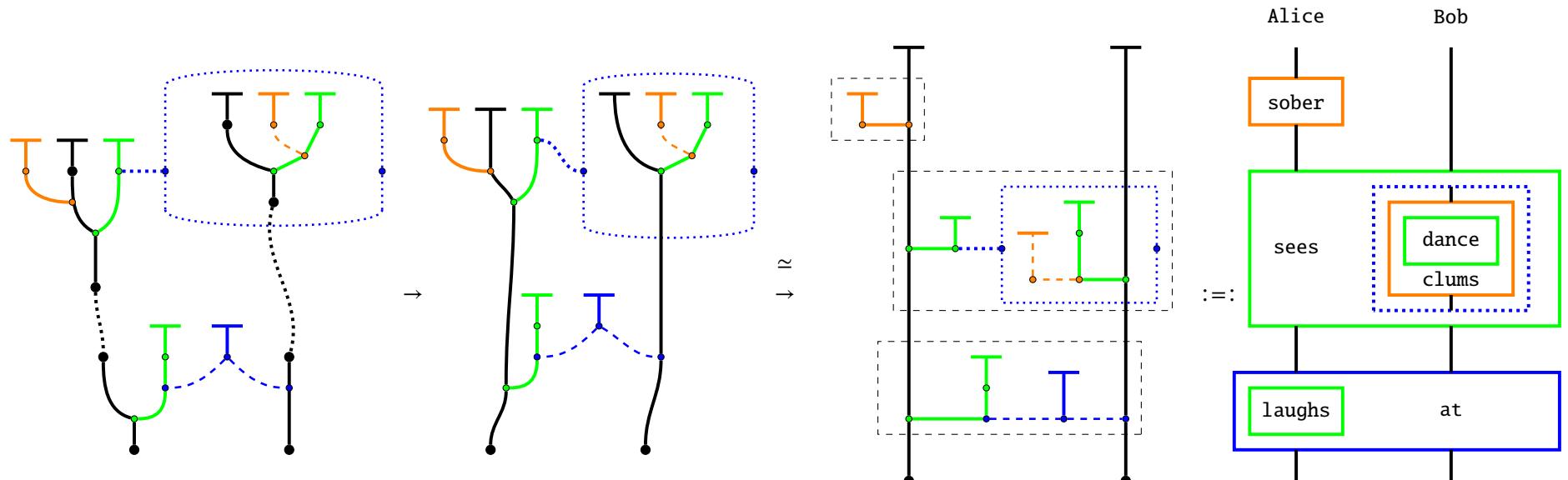
Rules 3.2.9 (Text to circuit). We turn finished text diagrams into text circuits by operating *in situ*, with extra rules outside the grammatical system that handle connecting noun wires.



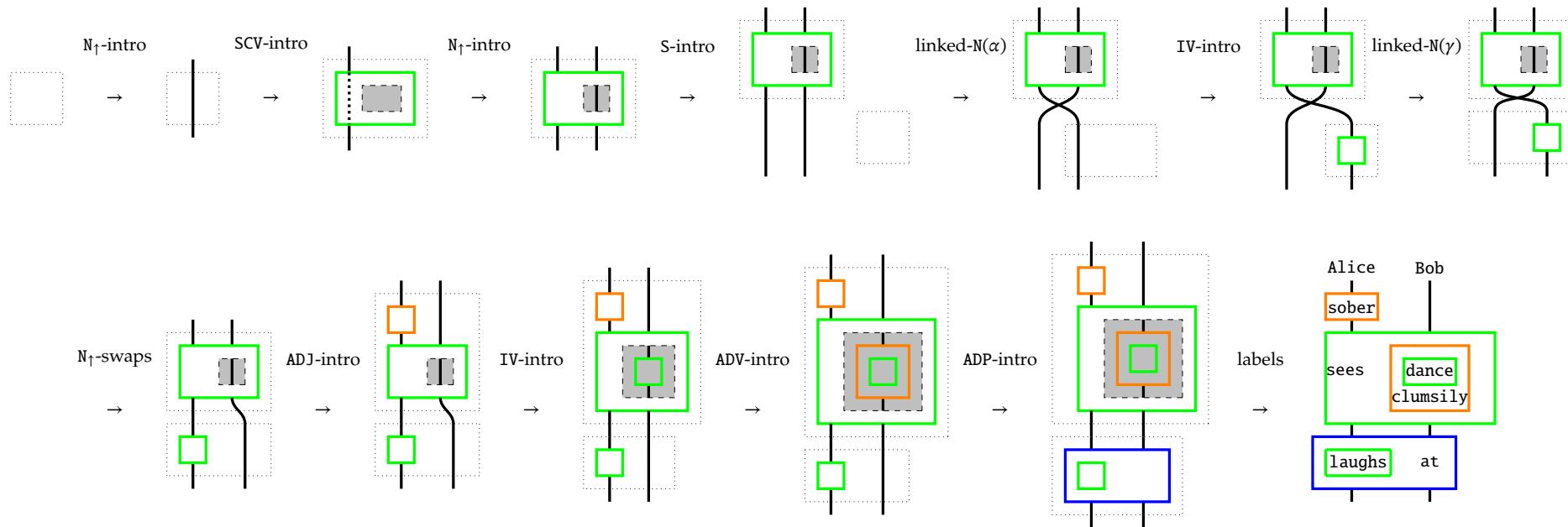
Example 3.2.10 (Text to circuit in action). In the first step below, by Lemmas 3.2.16 and 3.2.17, we can always rearrange a finished text diagram such that the noun wires are processive. In the second step, use the first rewrite of Construction 3.2.9 to prepare the wires for connection.



In the third step, we just ignore the existence of the bubble-scaffolding and the loose scalars. We could in principle add more rewrites to melt the scaffolding away if we wanted to. In the fourth step, we apply the second and third rewrites of Construction 3.2.9 to connect the wires and eliminate nodules underneath labels. We can also straighten up the wires a bit and make them look proper. At this point, we're actually done, because the resulting diagram *is already a text circuit up to a choice of notation*.



Example 3.2.11 (Growing circuits directly). Here is the combined content of Examples 3.2.4, 3.2.5, 3.2.8, this time presented directly as string-diagram rewrites treating text circuits as a primitive syntax, where dotted-boxes indicate sentence scopes.



The point is this: if it is possible to relate a string-diagram rewrite system to a linear syntax by graphical means, then (so I hope to have demonstrated) it may be done with a suitably specified weak n -category. So we may declare string diagram rewrite systems as we please and treat them as in-principle-formal generative grammars. I have demonstrated here how to construct a concrete witness for such a correspondence and to prove well-behaviour properties.

3.2.2 *Text circuit theorem*

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

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

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

Construction 3.2.12 (CSG for simple sentences). We may characterise simple sentences (containing only one verb) with the context-sensitive grammar in Figures 3.25 and 3.26.

Figure 3.25: Reading each diagram from top-to-bottom, from left-to-right we have generators for intransitive verbs, transitive verbs, adjectives, and adverbs. Generators for verbs require a number of N_\uparrow matching their arity as input, hence a CSG.

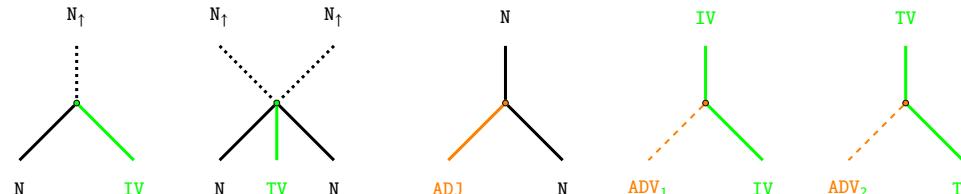
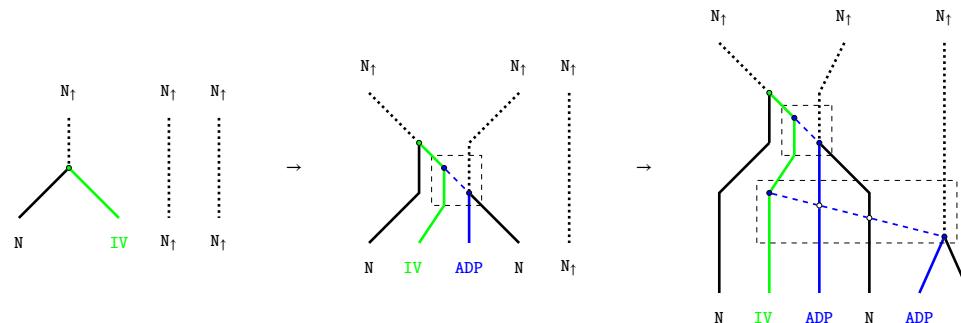


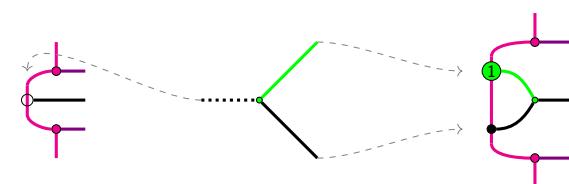
Figure 3.26: Adpositions require several helper-generators, which are the components within dashed boxes in the depicted example demonstrating the process of appending adpositions to an intransitive verb.



Proposition 3.2.13. Up to labels, the simple-sentence rules of the circuit-growing grammar are strongly equivalent to the CSG; in particular, they yield the same sentences.

Proof. By graphical correspondence between CSG rules and how the generators of the circuit-growing grammar changes surface nodes (Figure 3.27). \square

Figure 3.27: Viewing nodes on the pink surface of circuit-growing grammar as 1-cells, each rewrite rule yields a 2-cell; e.g. the dashed-blue helper lines for adpositions correspond to the ADP -pass rules in circuit-growing grammar. The correspondence between the IV -intro rules of both grammars is depicted.



Proposition 3.2.14. Up to labels, Rules 3.2.2 and 3.2.3 for simple and complex sentences yield the same sentences as the combined CSG of Construction 3.2.12 with the additional rules depicted in Figure 3.28.

Proof. Same correspondence as Proposition 3.2.13, ignoring the dotted-blue guards. \square

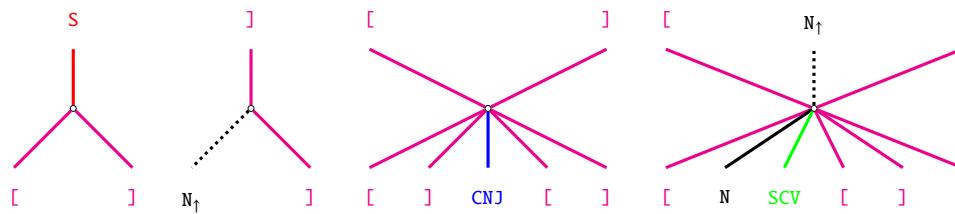


Figure 3.28: The first rule instantiates the left and right boundaries of a sentence, corresponding the starting bubble in circuit-growing grammar. The second corresponds to N_\uparrow -intro, the third CNJ -intro, and the fourth SCV -intro.

Now we address complex sentences and text by connecting nouns (Figure 3.29). This presents no issue across distinct simple sentences, but a complication arises when connecting nouns within the same simple sentence with reflexive pronouns e.g. Alice likes herself. Reflexive coreference would violate of the processivity condition of string diagrams for symmetric monoidal categories. Not all symmetric monoidal categories possess the appropriate structure to interpret such reflexive pronouns, but we several options exist, explored in Figure 3.30.

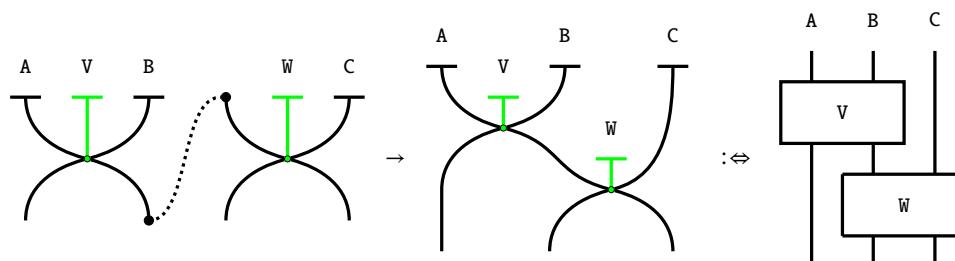
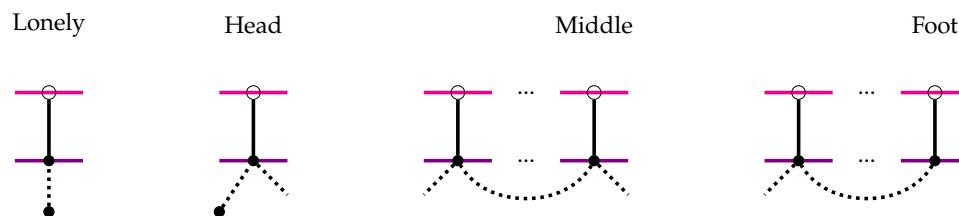


Figure 3.29: We choose the convention of connecting from left-to-right and from bottom-to-top, so that we might read circuits as we would English text: the components corresponding to words will be arranged in the reverse order, left-to-right and top-to-bottom.

Terminology 3.2.15 (Kinds of nouns with respect to coreference).



We classify kinds of nouns by their tails. *Lonely* nouns have no coreferences, their tails connect to nothing. *Head* nouns have a forward coreference in text; they have two tails, one that connects to nothing and the other to a noun later in text. *Middle* nouns have a forward and backward coreference; they have two tails, one that connects to a noun in some preceding sentence, and one that connects forward to a noun in a succeeding sentence. *Foot* nouns only have a backward coreference; they have a single tail connecting to a noun in some preceding sentence.

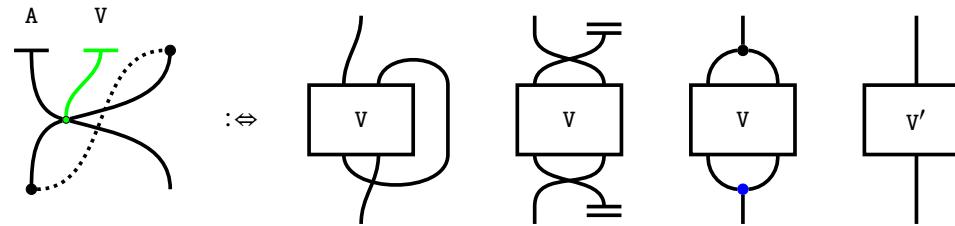
Lemma 3.2.16. The unsaturated nouns in Terminology 3.2.15 are exhaustive, hence nouns that share a coreference are organised as a diagrammatic linked-list.

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

Lemma 3.2.17. No nouns within the same sentence are coreferentially linked.

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

Figure 3.30: From left to right in roughly decreasing stringency, compact closed categories are the most direct solution for reflexive pronouns. Traced symmetric monoidal categories also suffice. So long as the noun wire possesses a monoid and comonoid, a convolution works. We also can just specify a new gate. We provide a purely syntactic treatment in [WLC23]; for now we treat them as if they were just verbs of lower arity.



Definition 3.2.18 (Finished text diagram). The circuit-growing grammar produces *text diagrams*. We call a text diagram *finished* if all surface nodes are labelled.

Proposition 3.2.19. Finished text diagrams yield text, up to interpreting distinct sentences as concatenated with punctuation `,`, `,`, contentless conjunctions or complementisers – such as `and`, `or` `that` respectively.

Proof. Sentence-wise grammaticality is gauged by Propositions 3.2.13 and 3.2.14. When multiple sentences occur within the scope of a SCV we might prefer the use of contentless complementisers and conjunctions, e.g. Alice sees that Bob draws and Charlie drinks , and Dennis dances _ is grammatically preferable but meaningfully equivalent to Alice sees (Bob draws Charlie drinks) Dennis dances. For our purposes it makes no difference whether surface text has these decorations, as the deep structure of text diagrams encodes all the information we care to know. \square

Proposition 3.2.20 (Finished text diagrams yield unique text circuits (up to processive isotopies)). *Proof.*

Every sentence corresponds to a gate up to notation, and we have a handle on sentences via Propositions 3.2.13 and 3.2.14. Lemmas 3.2.16 and 3.2.17 guarantee processivity. Uniqueness-up-to-processive-isotopy is inherited: text diagrams themselves are already specified up to connectivity, which entails equivalence up to processive isotopy. Therefore, for any circuit C obtained from a text diagram T by Construction 3.2.9, T can be modified up to processive-isotopy on noun wires to yield T' and another circuit C' that only differs from C up to processive isotopy, and all C' may be obtained in this way. \square

The converse of Proposition 3.2.20 would be that any text circuit that can be formed by the composition of symmetric monoidal categories and of plugging gates into boxes yields a text diagram. This would mean that text circuit composition is acceptable as a generative grammar for text. Establishing this converse requires elaboration of some conventions.

Convention 3.2.22 (Wire twisting). Wires are labelled by nouns. We consider two circuits the same if their gate-connectivity is the same. In particular, this means that we can eliminate unnecessary twists in wires to obtain diagrammatically simpler representations (Figure 3.31).

Convention 3.2.23 (Arbitrary vs. fixed holes). Diagrammatically, adverbs and adpositions are depicted with no gap between the bounding box and their contents, whereas conjunctions and verbs with sentential complement are depicted with a gap; this is a visual indication that the former are type-sensitive, and the latter can take any circuit.

Convention 3.2.24 (Sliding). Since only gate-connectivity matters, we consider circuits the same if all that differs is the horizontal positioning of gates composed in parallel (Figure 3.32).

Remark 3.2.21. There are some oddities about our conventions that will make sense later when we consider semantics. For example, Convention 3.2.22 an acceptable thing to ask for syntactically but quite odd to think about at the semantic level, where we would like to think that distinct nouns manifest as different states on the same noun-wire-type. A semantic interpretation that makes use of this convention will become clearer later in Sketch 5.5. Similarly, Convention 3.2.27 wouldn't be true if we consider the order of text to reflect the chronological ordering of events, in which case there are implicit `...` and `then ...` conjunctions that distinguish ordered gates from parallel gates conjoined by an implicit `...` `while ...`. This and the distinction in Convention 3.2.23 between typed and "untyped" higher-order processes will be given a suitable semantic interpretation in Sketch 5.4.

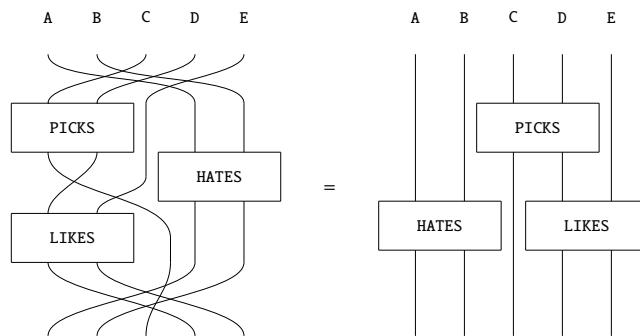


Figure 3.31: Only connectivity matters in text circuits, which we may use to freely rearrange and simplify presentations.

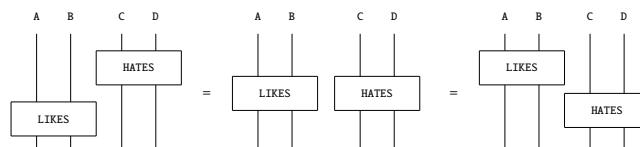
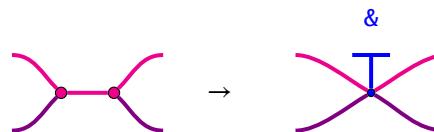


Figure 3.32: While sequential composition in process theories often has implicit temporality, this is not necessarily the case for text circuits, which may just (for instance) represent relational constraints. Temporality may be achieved in text circuits by interpreting them in premonoidal settings [Jef98], at the cost of the interchange rule depicted here.

Convention 3.2.25 (Reading text circuits). Pronominal connection conventions are to be chosen so that text circuits may be read in the same directional reading conventions of the language they aim to represent.

Convention 3.2.26 (Contentless conjunctions). Conventions 3.2.24 and 3.2.25 require something else to allow them to work at the same time: something to distinguish when the gates are parallel. In terms of text diagrams, we want rewrites that introduce such contentless conjunctions and witnesses their associativity, as in Figure 3.33:

Figure 3.33: Parallel gates represent compound sentences with contentless conjunctions. In English, some examples might be a punctuation mark such as a comma, or phrases such as and also.



Convention 3.2.27 (Lonely wires). There's only a single kind of text circuit we can draw that does not obviously correspond to a text diagram, and that's one where gates are missing (left). In process theories, wires are identity processes that do nothing to their inputs. So to deal with lonely wires in terms of text diagrams, we want a rewrite that introduces such contentless verbs (Figure 3.34).

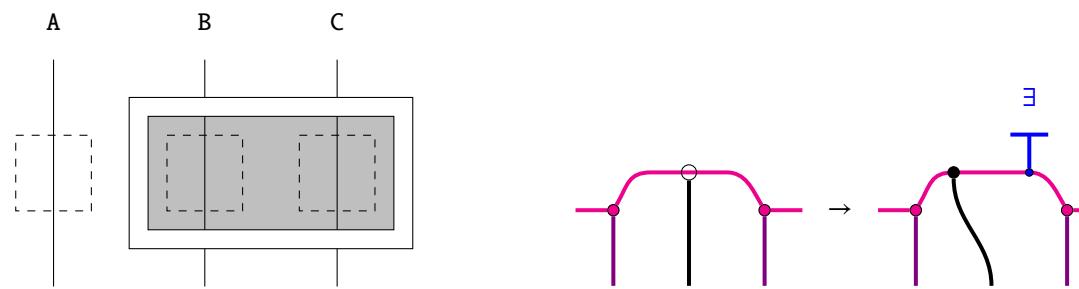
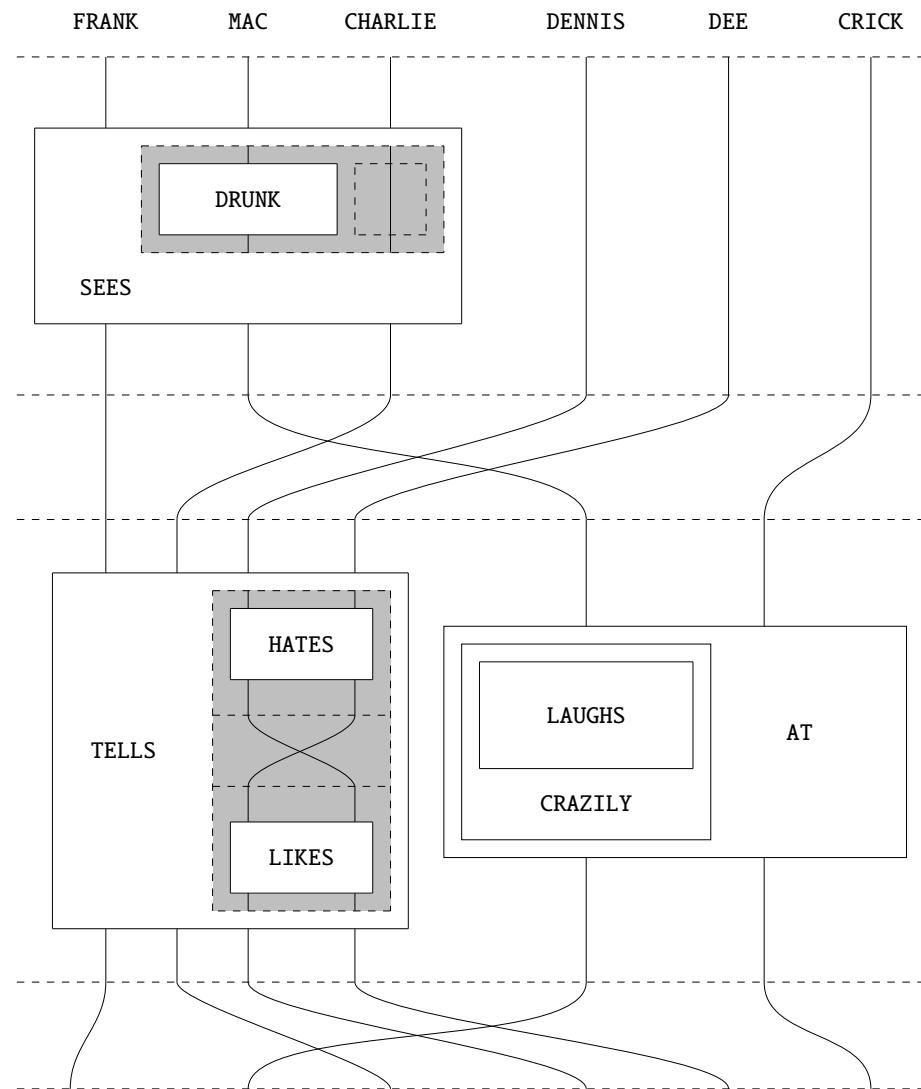
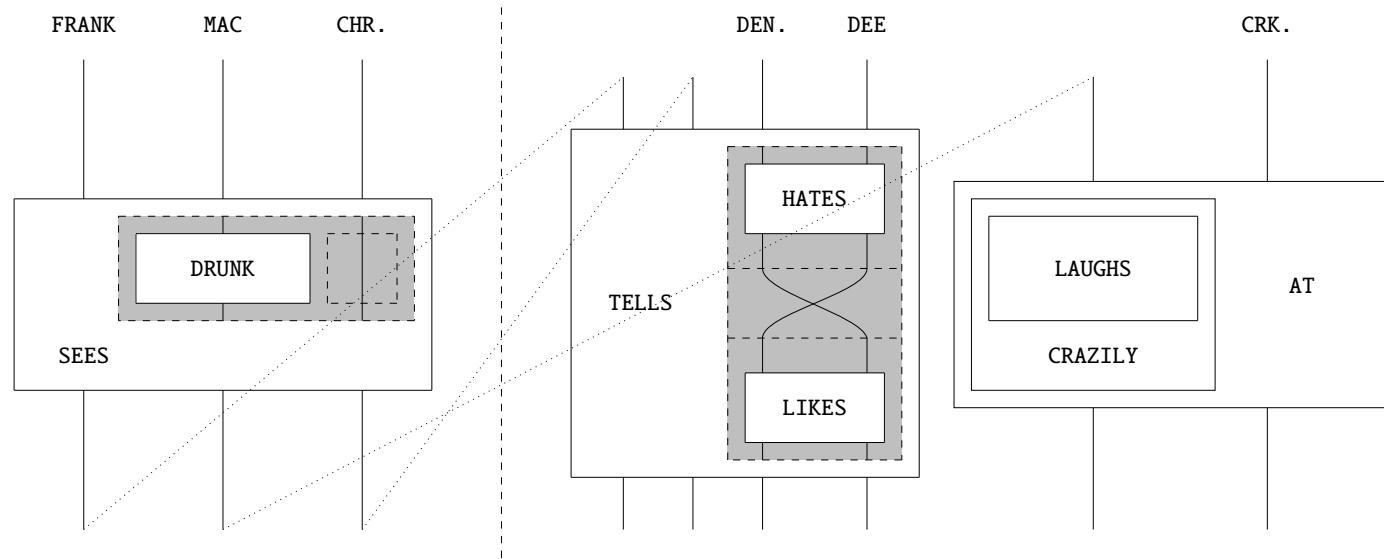


Figure 3.34: Lonely wires in text circuits are identity processes. We require a text diagram analogue, and an intransitive "null-verb" in English that seems to work is *is*, in the sense of exists.

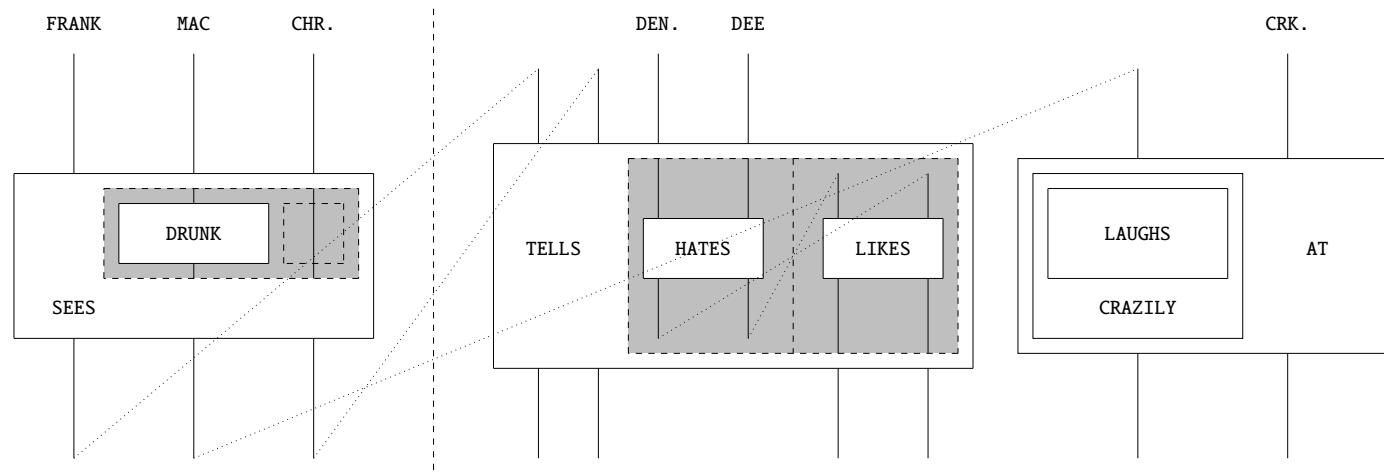
Construction 3.2.28 (Circuit to text). In the presence of additional rewrites from Conventions 3.2.26 and 3.2.27, every text circuit is obtainable from some text diagram, up to Conventions 3.2.22 and 3.2.24. Starting with a circuit, we may use Convention 3.2.22 to arrange the circuit into alternating slices of twisting wires and (possibly tensored) circuits, and this arrangement recurses within boxes. Slices with multiple tensored gates will be treated using Convention 3.2.26. By convention 3.2.27, we decorate lonely wires with formal `exists` gates, as in the `Frank sees` box. Observe how verbs with sentential complement are depicted with grey gaps, whereas the adverb and adposition combination of `Mac crazily laughs at Cricket` is gapless, according to Convention 3.2.23.



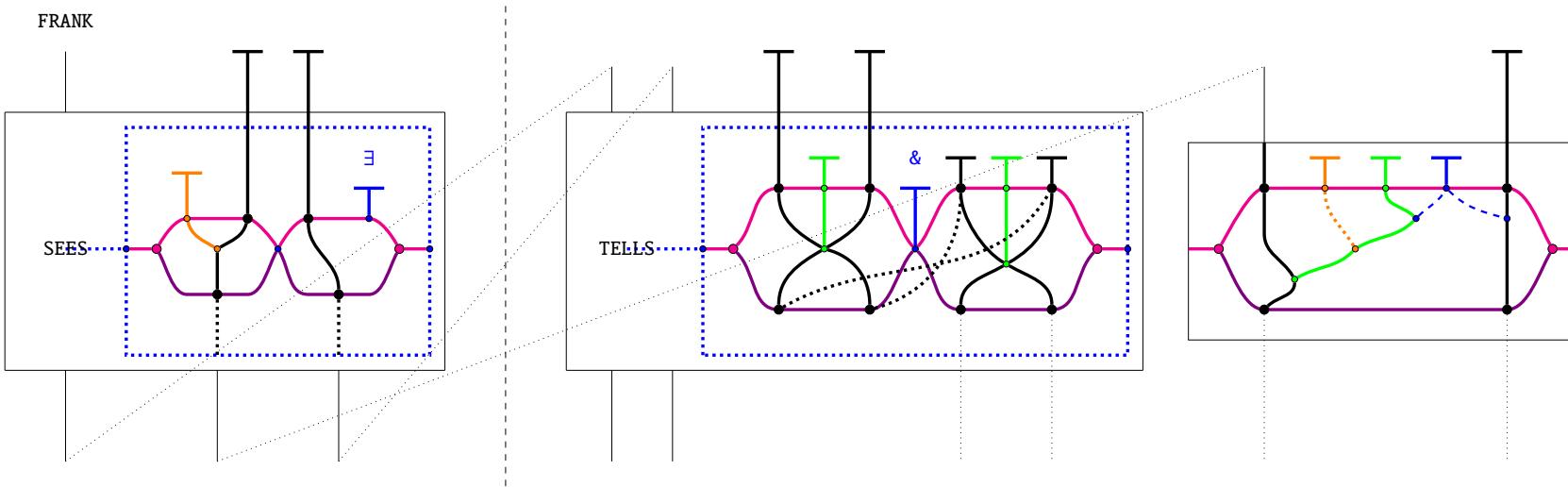
We then linearise the slices, representing top-to-bottom composition as left-to-right. Twist layers are eliminated, replaced instead by dotted connections indicating processive connectivity. The dashed vertical line distinguishes slices. This step of the procedure always behaves well, guaranteed by Proposition 3.2.16. Noun wires that do not participate in earlier slices can be shifted right until the slice they are introduced.



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

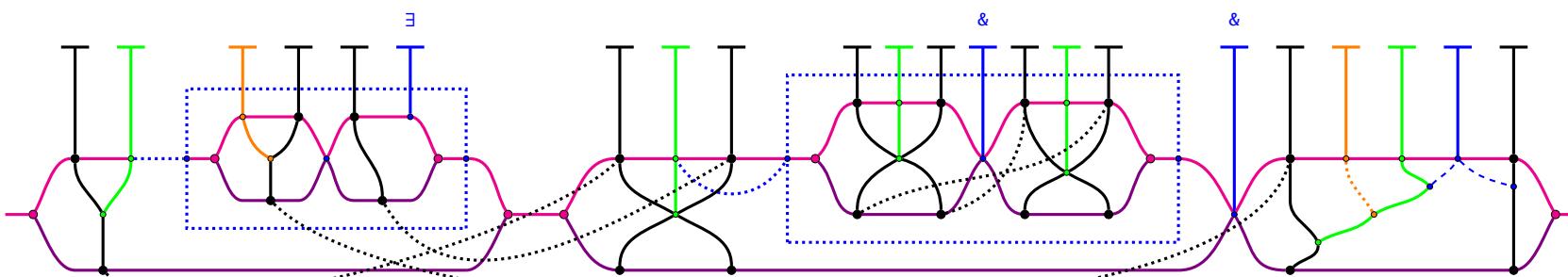


By Proposition 3.2.14, gates are equivalent to sentences up to notation, so we swap notations *in situ*. Conventions 3.2.26 and 3.2.27 handle the edge cases of parallel gates and lonely wires. Observe that the blue-dotted wiring in text diagrams delineates the contents of boxes that accept sentences.



Recurring notation swaps outwards and connecting left-to-right slices as sentence-bubbles connect yields a text circuit, up to the inclusion of rewrites from Conventions 3.2.26 and 3.2.27: applying the reverse of those rewrites and the reverse of text-diagram rewrites yields a valid text-diagram derivation, by Propositions 3.2.14 and 3.2.16. We haven't formally included transitive verbs with sentential complement in our vocabulary, but it should be obvious at this point how they function with our existing machinery.

Frank sees [drunk Mac (&) Charlie (\exists)].
 Frank tells Charlie [Dennis hates Dee (&) Dee likes Dennis]
 (&) Mac crazily laughs at Cricket.



Theorem 3.2.29 (Text Circuit Theorem). Text generated by the circuit-growing grammar is sensible and surjects onto text circuits. Hence:

Text circuits are a generative grammar for text

Proof. Sensibility at the sentential level is established by Propositions 3.2.13 and 3.2.14. Proposition 3.2.20 establishes a map from text to circuits, and Construction 3.2.28 witnesses its surjectivity. The conventions required for the construction are accompanied by justifications of sensibility. \square

An example from Task 1, "single supporting fact", is:

Mary went to the bathroom.

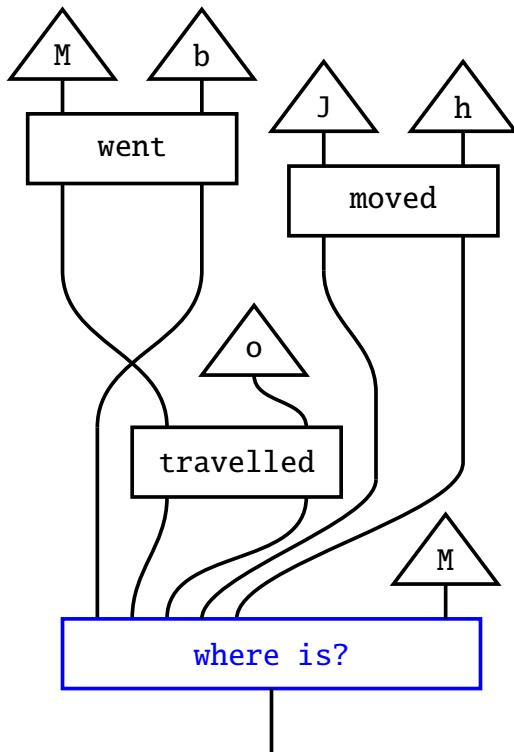
John moved to the hallway.

Mary travelled to the office.

(Query:) Where is Mary?

(Answer:) office.

Translating the setup of each task into a circuit of neural nets-to-be-learnt, and queries into appropriately typed measurements-to-be-learnt, each bAbi task becomes a training condition in the form of a process-theoretic equation to be satisfied: the depicted composite process ought to be equal to the office state:



3.3 Text circuits: details and development

This section covers some practical developments, references for technical details of text circuits, and sketches of how to play with them. The most striking demonstration to date is that circuits are defined over a large enough fragment of language to *leverage* several bAbi tasks [BAb], which are a family of 20 general-reasoning text tasks – the italicised choice of wording will be elaborated shortly. Each family of tasks consists of tuples of text in simple sentences concluded by a question, along with an answer. It was initially believed that world models were required for the solution of these tasks, but they have been solved using transformer architectures. While there is no improvement in capabilities by solving bAbi using text circuits, the bAbi tasks have been used as a dataset to learn word gates from data, in a conceptually compliant and compositional manner, detailed in the margin. Surprisingly, despite the low-data, low-compute regime, the tasks for which the current theory has the expressive capacity to cover are solved better by text circuits than by LSTMs; a proof-of-concept that with the aid of appropriate mathematics, not only might fundamental linguistic considerations help rather than hinder NLP, but also that explainability and capability are not mutually exclusive. Experimental details are elaborated in a forthcoming report [ano]. While there are expressivity constraints contingent on theoretical development, this price buys a good amount of flexibility within the theoretically established domain: text circuits leave room for both learning-from-data and "hand-coded" logical constraints expressed process-theoretically, and naturally accommodate previously computed vector embeddings of words.

In practice, the process of obtaining transparently computable text goes through two phases. First, one has to obtain text circuits from text, which is conceptually simple: typological parsers for sentences can be modified to produce circuit-components rather than trees, and a separate pronomial resolution module dictates symmetric monoidal compositionality; details are in the same forthcoming report. Second, one implements the text circuits on a computer. On quantum computers, boxes are modelled as quantum combs. On classical computers, boxes are sandwiches of generic vector-manipulation neural nets, and boxes with 'dot dot dot' typing are interpreted as families of processes, which can be factored for instance as a pair of content-carrying gates along with a monoid+comonoid convolution to accommodate multiplicity of wires; an example of this interpretation of families of processes is the use of an aggregation monoid in graph neural network [DV22]. The theoretical-to-practical upshot of text circuits when compared to DisCoCat is that the full gamut of compositional techniques, variations, and implementation substrates of symmetric monoidal categories may be used for modelling, compared to the restrictions inherent in hypergraph and strongly compact closed categories.

In terms of underpinning mathematical theory, the 'dot dot dot' notation within boxes that indicates related families of morphisms is graphically formal [WGZ22], and interpretations of such boxes were earlier formalised in [Mer14, Qui15, Zam17]. Boxes with holes may be interpreted in several different ways. Firstly, boxes may be considered syntactic sugar for higher-order processes in monoidal closed categories, and boxes

are diagrammatically preferable to combs in this regard, since the latter admits a typing pathology where two mutually facing combs interlock. Secondly, boxes need not be decomposable as processes native to the base category, admitting for instance an interpretation as elementwise inversion in linear maps, which specialises in the case of **Rel** (viewed as **Vect** over the boolean ring) to negation-by-complement. In some sense, none of these formalities really matter, on the view that text circuits are algebraic jazz for computing with text, where facets are open to interpretation and modification.

3.4 How to play

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

FUNCTIONAL WORDS. No separate "information structure" is required; all is transparent and manipulable. For instance, in the case of relative pronouns in Example 3.4.2, ansatz wires may be freely introduced and manipulated to achieve the desired correspondence between deep and surface structure.

GRAMMAR EQUATIONS. A principle for the extension of text diagrams to larger fragments of text is to devise *grammar equations* [CW21] that express novel textual elements in terms of known text diagrams, for instance, that the use of passive voice or copulas amounts to swaps in the linear surface order, as in Examples 3.4.3 and 3.4.4. Syntactic rules may even introduce semantic components, as in for instance the treatment of the possessive pronoun in Example 3.4.5. In sum, intuitions about the systematicity of language are directly translatable into rewrite rules that manipulate deep structure as one sees fit.

GRAMMATICAL TYPING IS NESTING. A heuristic for the extension of text diagrams to novel grammatical categories is the absorption of type-theoretical compositional constraints at the level of sentences into nesting of boxes, as demonstrated in Examples 3.4.6 and 3.4.7.

SYNCGATEOREMATICITY. A useful heuristic for the application of text diagrams is to treat individual text circuits as analogous to propositional contents, and certain logical or temporal connectives as structural operations upon circuits – rewrites – that must be applied in order to obtain a purely propositional format. In other words, logical or structural words are to be treated as circuit-manipulation instructions to be executed in order to obtain a circuit, in the same way that $1 + 1$ is only an integer expression once addition has been evaluated. See Examples 3.4.8 and 3.4.9 for a demonstration.

SYNTAX IN CONTEXT. Extending the reach of text circuits to determiners, quantifiers, and conditionals appears

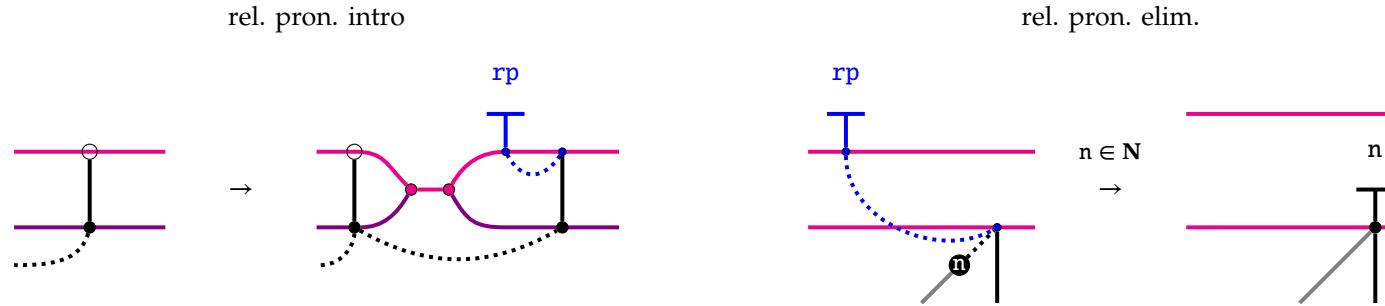
to require a contextual process theory in which to evaluate and enforce constraints upon the purely syntactic content of text circuits. The broad strategy sketched here rests upon three tactics. First, as in the neural approach to bAbi, word-gates are considered to be paired with measurement-processes that return an analog of truth values, the latter of which may be generic tests for adjectives as static predicates or verbs as dynamic predicates. The pairing of gates with measurements follows the philosophy of update structures [HWW20]. The truth-measurements allow conditionals to be expressed as either circuit-rewrites or constraints on truth-measurements, the latter which are in turn interpretable as loss-functions in the process of training gates. Second, we model context as the rest of the text circuit, which is a modifiably finite model. Third, we suppose we have a way to record and relate alternative circuits. These tactics appear sufficient for a first pass. Determiners may be considered to be context-sensitive connectivity. Universal quantifiers may be analysed in particular finitary contexts as conditionals and constraints on truth-conditional measurements. Existential quantifiers evaluated in the finitary case yield alternative circuits. See Examples 3.4.11, 3.4.12, 3.4.13, 3.4.15, and 3.4.15 for demonstrations.

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

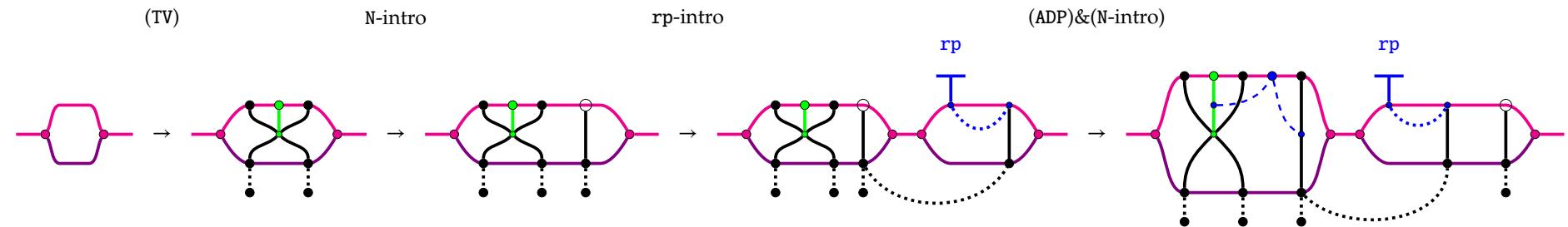
OBJECTION!: HOLD ON, ISN'T THIS JUST TRANSFORMATIONAL GRAMMAR? HAVEN'T WE MOVED ON FROM THAT? In spirit, yes. The two major mathematical distinctions here are well-typing and many-input-many-output instead of treelike. The practical distinction is that this theory works with neural nets. Both approaches have the same theoretical problems: over- and undergeneration, no evidentiary basis for psychological realism, too rigid for functionalists, and so on. But recall that we differ in aims: our formalist approach is ultimately in service of approximating human language structure in machines for interpretability. How so? Solving language tasks such as bAbi via text circuits also means that each word gate has been learnt in a conceptually-compliant manner, insofar as the grounded meanings of words are reflected in how words interact and modify one another. What is meant by "conceptually-compliant" is a stronger variant of Firth's maxim: "the meaning of a word is how it interacts with other words". How do we justify that claim? The initial conception of bAbi was that the ability to answer questions about – for instance, the verb *to go* – in many different contexts

amounted to having a consistent internal "world-model". But question-answering performance by itself is evidently insufficient for the degree of interpretability implied by conceptual-compliance, because the internal model is not forthcoming in transformer solutions. On the other hand, we *do* obtain the building blocks of compositional world-models by learning word gates in text circuits: each learnt word gate may be considered a well-grounded semantic primitive in the construction of novel text circuits, and the resulting circuits are modifiable world-models that are queriable using the (also learnt) measurement-gates. Why is that so? Because just as in Section 1.6 we do not need to know how an update is implemented if it satisfies characteristic operational constraints imposed by process-theoretic equations, we don't need to know what's going on inside the gate to go so long as it satisfies the process-theoretic equations that to go ought to satisfy. What are these equations? Firth says that it is how to go behaves with respect to all other words in all contexts, which we approximate by translating individual bAbi tasks involving the word to go, via text circuits, into a representative sample of the process-theoretic equations that to go ought to satisfy. So the philosophical strength of the claim that to go and synonyms have been learnt-from-data in a way that coheres with human conceptions rests on three points: performance, Firth (or if you like, the Yoneda Lemma), and the breadth and variety exhibited in the bAbi dataset. The real test is practical demonstration, for which time will tell.

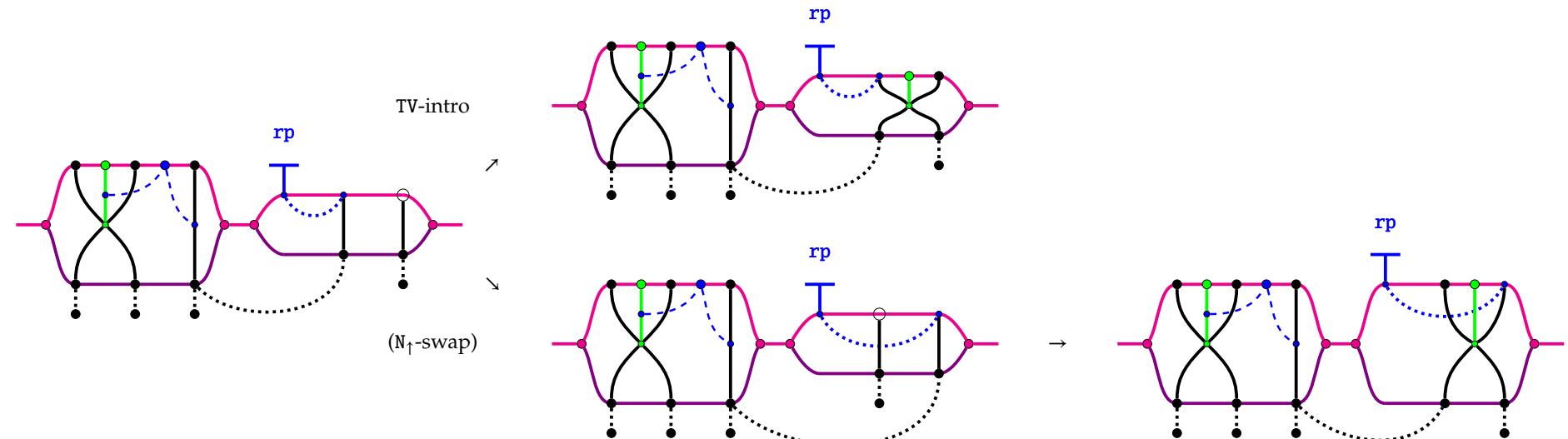
Rules 3.4.1. A relative pronoun glues two sentences across a single noun. For example, what would be a text with two sentences Alice teaches at school. School bores Bob. can be composed by identifying school: Alice teaches at school that bores Bob. In this case, that is called a subject-relative pronoun, as it stands in for the subject argument in bores. In Alice teaches at school that Bob attends., we have an object-relative pronoun, as that stands in for the object argument in attends. In English, relative pronouns occur immediately after the noun they copy, and are followed immediately by a sentence missing a noun. Reflexive pronouns may be dealt with in a similar fashion as we have with relative pronouns, and various semantic interpretation options have been covered already in Figure 3.30. We can extend our system to accommodate relative pronouns as follows. The introduction of a relative pronoun is considered to start a new sentence with a premade pronominal link. The relative pronoun connects to a new kind of surface noun, which for most intents and purposes behaves just like the nouns we have seen before, except for two caveats: it cannot be labelled (since the relative pronoun is already handling that), and it cannot leave the sentence it starts in by N-shift rules. We can eliminate relative pronouns by forcing the governed noun to be named, which is equivalent to dropping the relative pronoun and recovering distinct sentences.



Example 3.4.2 (Introducing relative pronouns). Here we demonstrate derivations of Alice teaches at school that bores Bob and Alice teaches at school that Bob attends. The initial steps in both cases are the same, setting up the teaches phrase structure and introducing a new unsaturated noun in the Bob phrase to work with the relative pronoun.



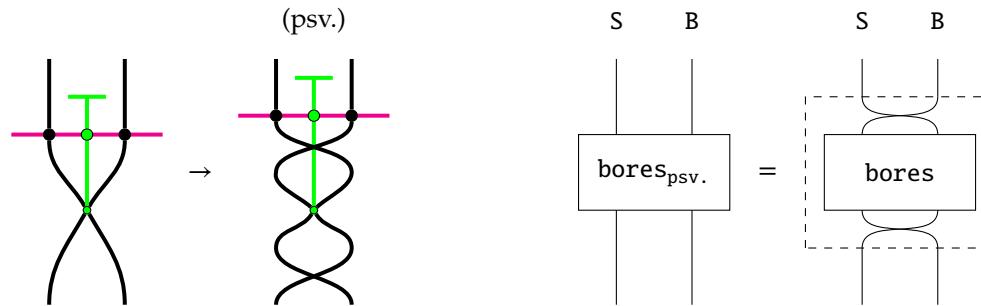
Now have a branching derivation. We may either directly generate a transitive verb treating the relative pronoun as a subject, or we may first perform an N_{\uparrow} -swap first and then generate a transitive verb, treating the relative pronoun as an object. Now the ends of either branch can be labelled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



Example 3.4.3 (Passive voice).

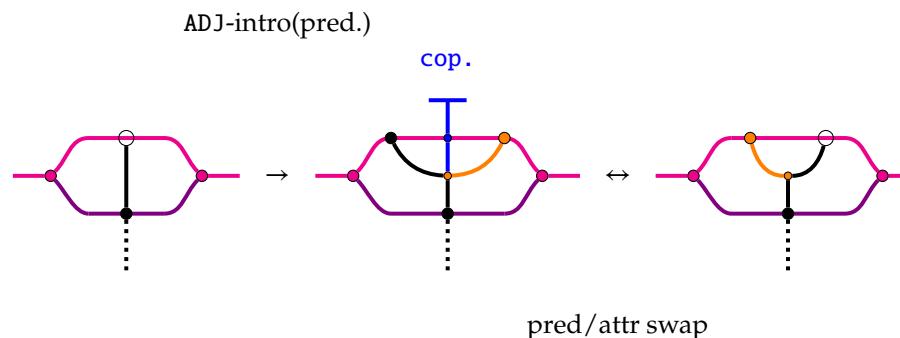
School bores Bob = Bob is bored by school

Twists in wires can be used to model passive voice constructions, which amount to swapping the argument order of verbs. In the original [WLC23], a more detailed analysis including the flanking words *is bored by* involves introducing a new diagrammatic region, which is modelled by having more than a single 0-cell in the n -categorical signature.

**Example 3.4.4 (Copulas).**

Red car = Car is red

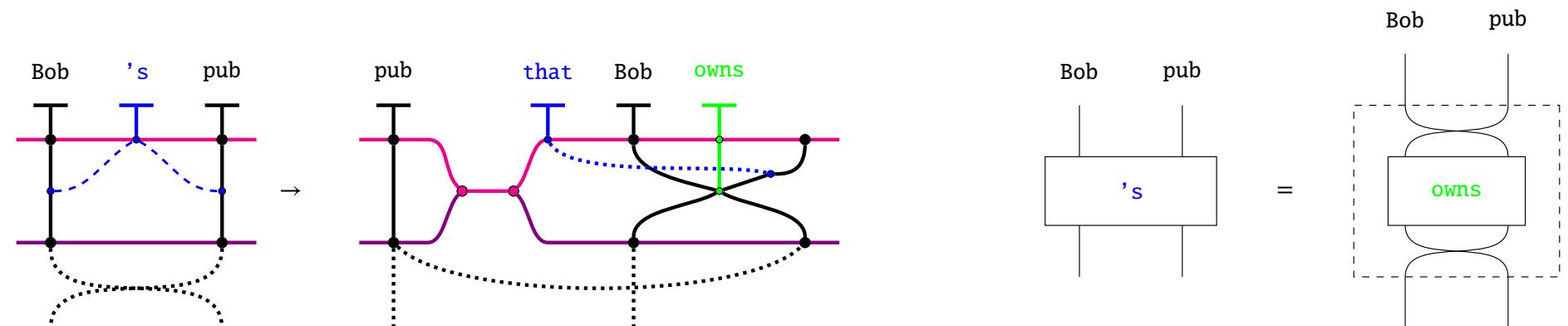
Modifiers such as adjectives and adverbs when they occur before their respective noun or verb are called *attributive*. When modifiers occur after their respective target, they are called *predicative*. In English, without the aid of and, only a single predicative modifier is permissible, e.g. *big red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



Example 3.4.5 (Possessive pronouns).

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

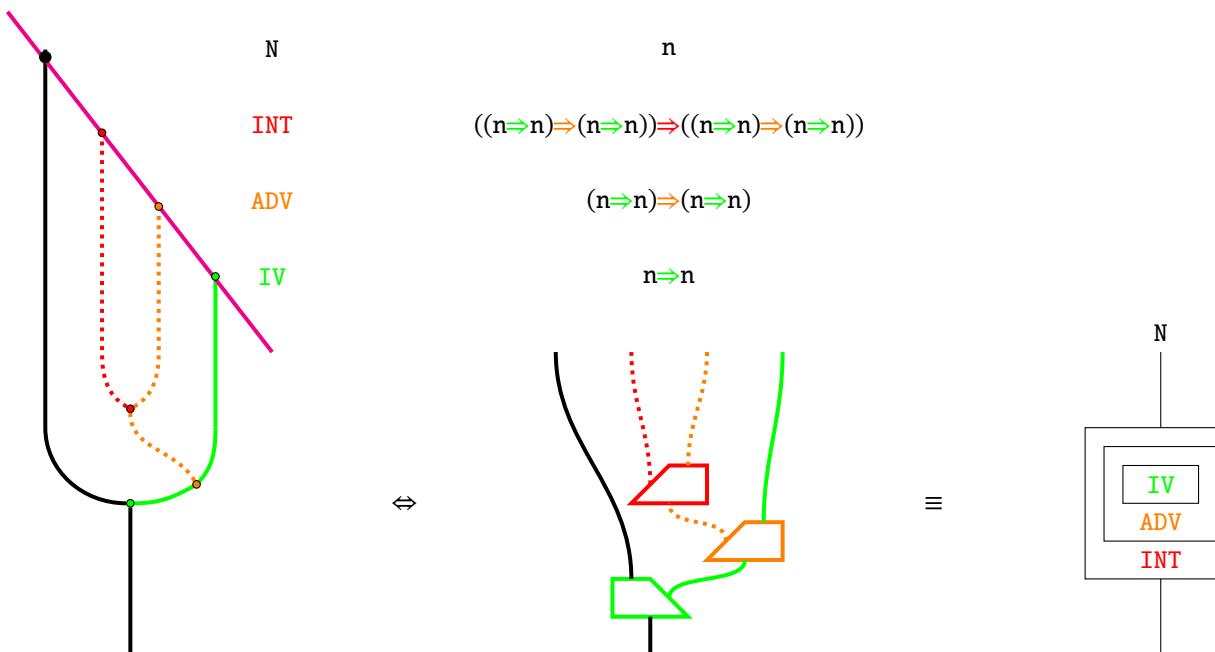
This example, along with other grammar equations, was first introduced in the pregroups and internal wirings context in [CW21]. Possessive pronouns are placed contiguously in between noun-phrases, for which the diagrammatic technology we developed for placing adpositions can be repurposed. Possessive pronouns may be dealt with by a single rewrite that relies on the presence of a transitive ownership verb in the lexicon, which corresponds to a box-analysis in text circuits.



Example 3.4.6 (Intensifiers).

Alice very quickly runs

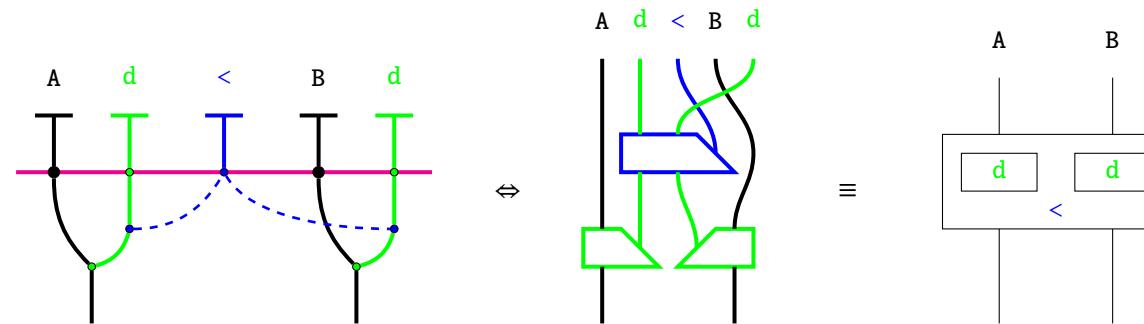
The deep nodes of a text diagram may be equivalently viewed as evaluators in a symmetric monoidal closed setting, and the surface nodes as states for the evaluators. By Curry-Howard-Lambek, this view recovers typological grammar settings where composition is some variant of modus ponens. So long as the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



Example 3.4.7 (Comparatives).

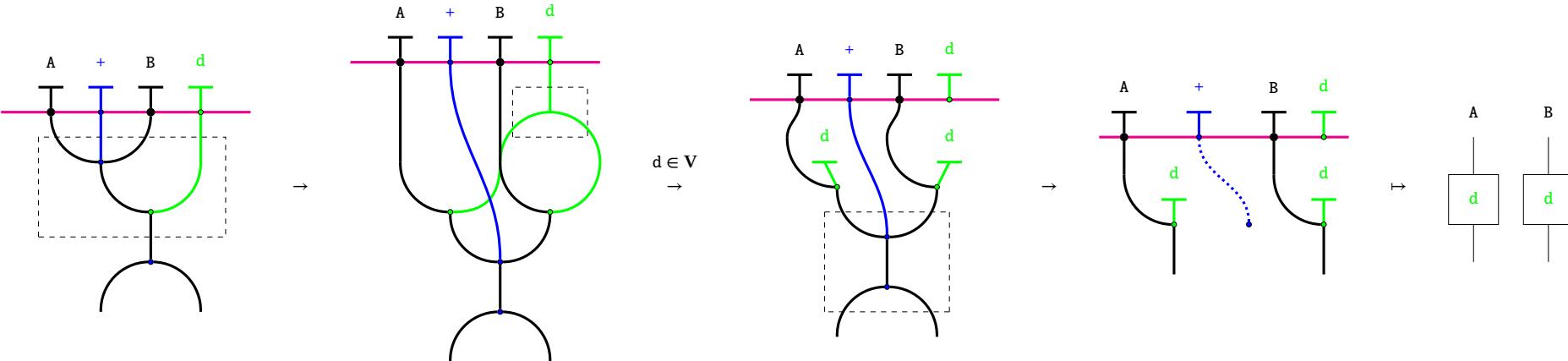
Alice drinks less than Bob drinks

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



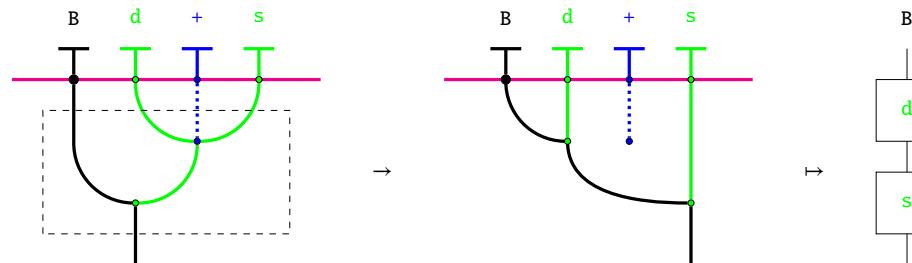
Example 3.4.8 (Syncategorematic I). Syncategorematic words are roughly those that have contextually-dependent semantics. Their dependency is usually predicated on the grammatical type of their arguments. In our terms, since we consider the semantics of text circuits to be underpinned by monoidal functors that reify the circuits in a target category, syncategorematic words such as and may be treated as distributive laws. Here and occurs as a conjunction of nouns and is eliminated by distributive-law rewrites within the deep structure of the text diagram *before translation into circuits*. Note that what is meant by *distributive* here is, in string-diagrammatic terms, precisely the same as that in algebra, for expressions such as $a \times (b + c) = (a \times b) + (a \times c)$. A new copy-node for verb labels that has rewrites for all verbs facilitates distribution, and the deep and nodes come in a tensor-dentensor pair analogous to those for nonstrict string diagrams. Sources of rewrites are outlined in dashed boxes.

Alice and Bob drink

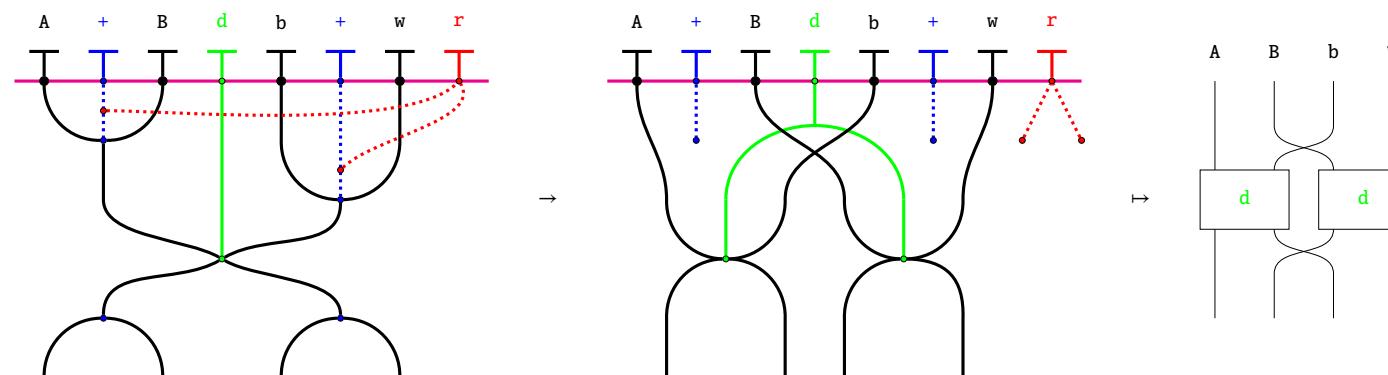


Example 3.4.9 (Syncategorematicity II).Bob drinks and smokes

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

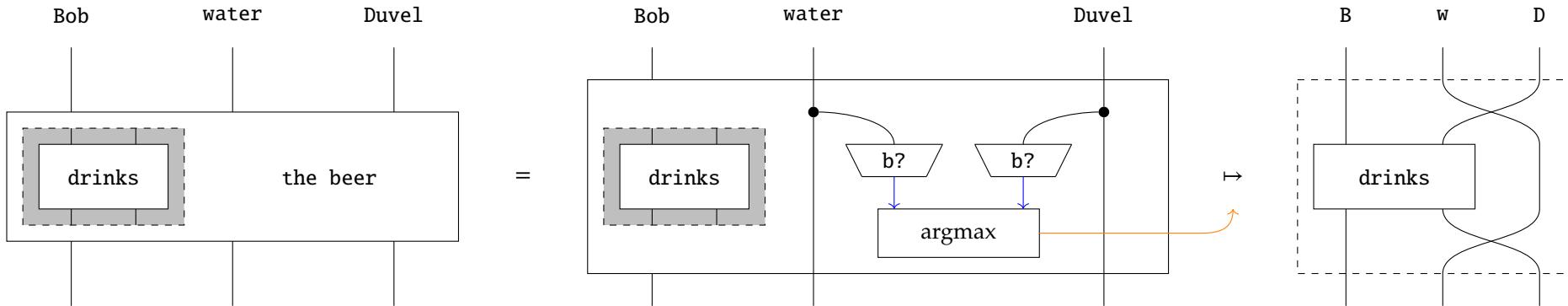
**Example 3.4.10 (Coordination).**Alice and Bob drink beer and wine respectively

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

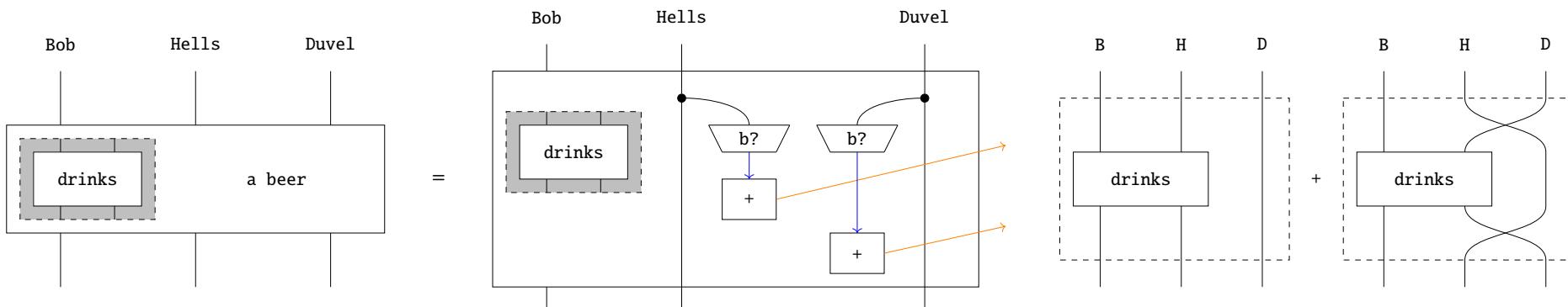


Example 3.4.11 (Determiners I).Bob drinks the beer (among drinks)

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

**Example 3.4.12 (Determiners II).**Bob drinks a beer (among drinks)

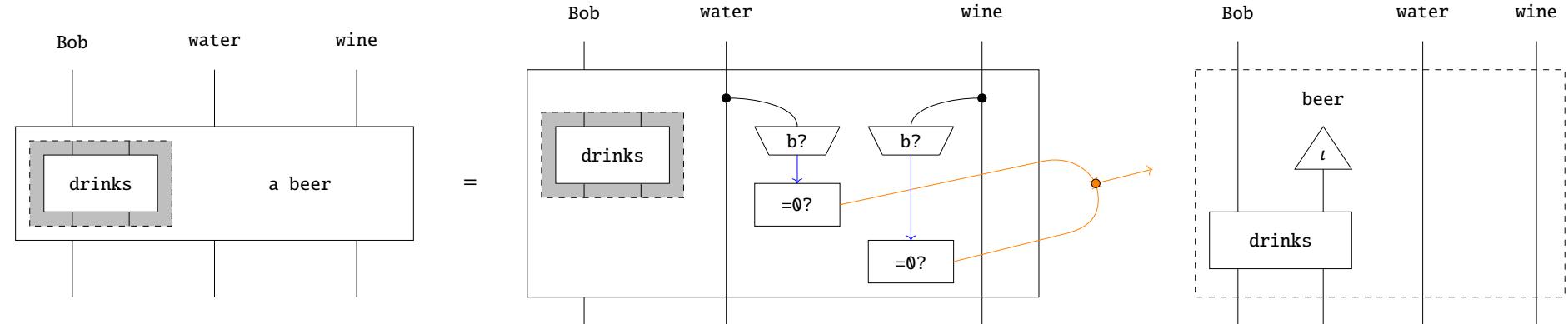
We take the logical (and pragmatic) reading of *a* as $\exists!x : \text{beer?}(x) \wedge \text{drinks?}(\text{Bob}, x)$. Subject to having a method to hold onto alternatives – in essence an inquisitive semantics approach – we may create alternative circuits for each successful *beer?* test.



Example 3.4.13 (Determiners III).

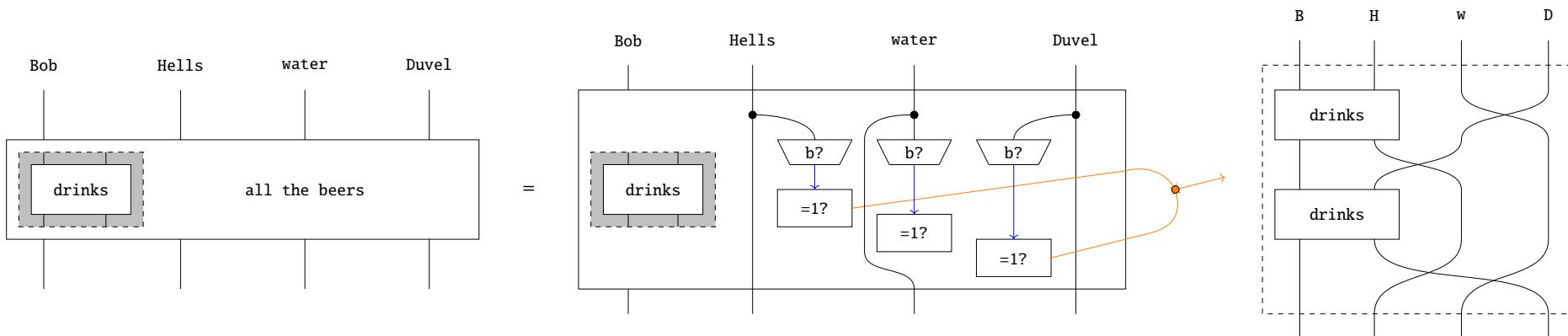
Bob drinks a beer (that we didn't know about)

When there are no beers in context, the same statement takes on a dynamic reading: it constitutes the introduction of a beer into discourse. In terms of text circuits, this amounts to introducing a novel beer-state and beer-wire. Determining an appropriate setting to accommodate "arbitrary" vs. "concrete" beers (c.f. Fine's arbitrary objects [Urq20]) requires further research and experimentation, but preliminarily it is known that density matrices are capable of modelling semantic entailment [BSC15], at the computational cost of adopting the kronecker product. This diagram doesn't typecheck, but note that it doesn't have to, because our strategy for evaluation of determiners treats circuits as syntactic objects to be manipulated.



Example 3.4.14 (Quantifiers I).Bob drinks all the beers (in context)

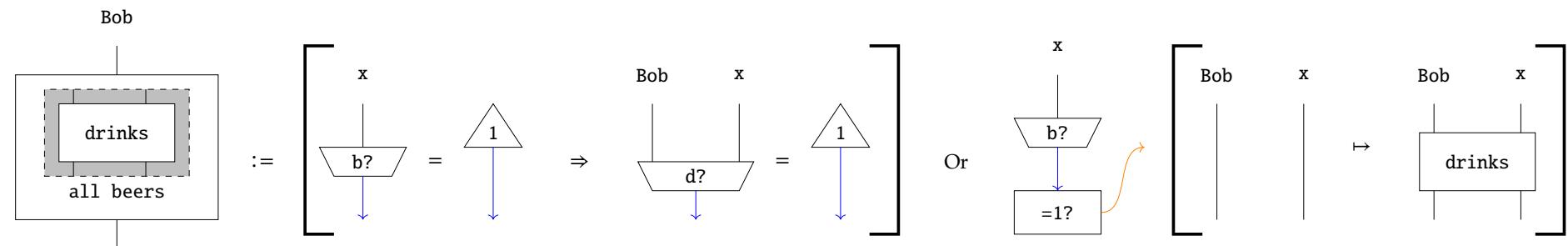
In a finitary context, drinking all the beers amounts to applying the distributivity of and iteratively in that context. In this case, all the beers is treated as a reference-in-context to Hells and Duvel. In the same manner, existential quantifiers in finite contexts can be treated as finitary disjunctions, which is handled by creating alternative circuits, as in Example 3.4.12



Example 3.4.15 (Quantifiers II).

Bob drinks all beers (generic)

Without the determiner **the**, this becomes a generic statement, which logically amounts to (analysing the usual conditional as a disjunction) $\forall x : \neg\text{beer?}(x) \vee \text{drinks?}(\text{Bob}, x)$. We can treat generic universal quantifiers of this kind in at least two ways. The first essentially truth-conditional approach is to treat the generic as a process-theoretic condition governing measurements: whenever it is the case that something is a beer, it is the case that Bob drinks it. The second "inferential" approach is to treat the generic as a rewrite of text circuits conditioned on a beer test: whenever something is a beer we may add on a gate witnessing that Bob drinks that beverage.



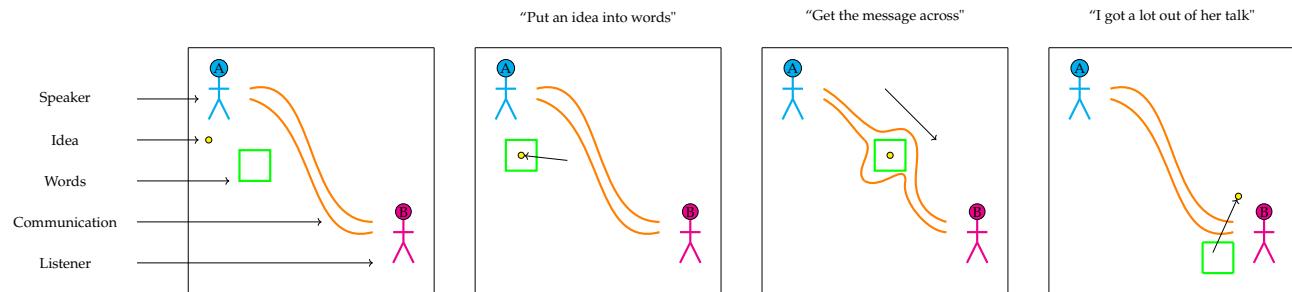
4

Continuous relations for semantics

We want a setting within which to reason formally with and about pictorial iconic representations, of the sort one might draw to illustrate a conceptual schema from cognitive linguistics. For this I introduce and investigate the category of continuous relations, **ContRel**.

4.1 Continuous Relations for iconic semantics

Figure 4.1: Sometimes it is very helpful to illustrate concepts using iconic representations in cartoons. For instance in the *conduit metaphor* [Red], words are considered *containers* for ideas, and communication is considered a *conduit* along which those containers are sent.



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

The main reason we prefer **ContRel** to either **Rel** or **Top** for our purposes is that we can diagrammatically characterise set-indexed collections of mutually disjoint open sets as *sticky-spiders*: a generalisation of spiders that interact with idempotents. We can then treat the indexing set as a collection of labels, and an indexed open set as a doodle. Notably, spiders don't exist in cartesian **Top** except for the one-point space, and the spatial structure of open sets doesn't exist in **Rel**. But there are all kinds of poorly behaved open sets even on the plane, so enter the next benefit: In **ContRel**, we can diagrammatically characterise the reals as a topological space up to homeomorphism, which gives us a diagrammatic handle on paths and homotopies, mathematical concepts that enable us to diagrammatically characterise when open sets are connected, how they might move and transform continuously in space, and when open sets are contained inside others. And once we've formalised doodles we'll be able to treat ourselves to cartoons as formal semantics for language and nobody can stop us.

SIDENOTE FOR CATEGORY THEORISTS

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

First, it is because and not despite of the naïvety 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 Kriesli 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.

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

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 $n \in \mathbb{N}$:

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

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

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

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

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

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

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 naively extend the notion of continuous maps to continuous relations between topological spaces.

Notation 4.2.4. For shorthand, we denote the topology (X, τ) as X^τ . As special cases, we denote the discrete topology on X as X^* , and the indiscrete topology X^0 .

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

4.2 Continuous Relations by examples

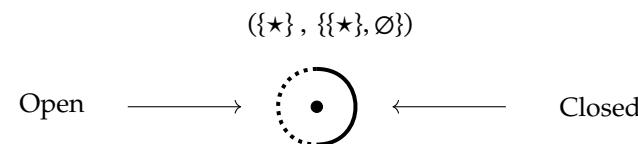
Definition 4.2.12 (Continuous Relation). A continuous relation $R : (X, \tau) \rightarrow (Y, \sigma)$ is a relation $R : X \rightarrow Y$ such that

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

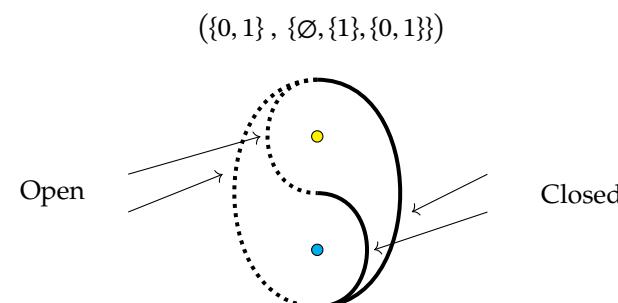
where \dagger denotes the relational converse.

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

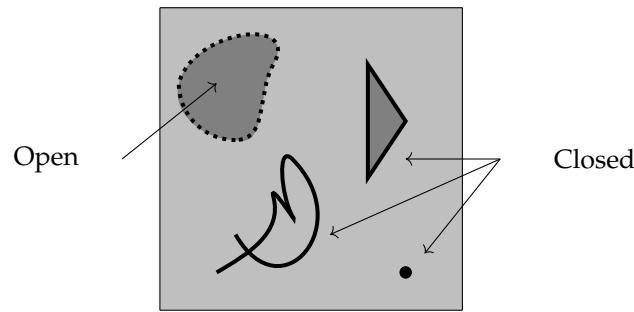
The **singleton space** consists of a single point which is both open and closed. We denote this space \bullet . Concretely, the underlying set and topology is



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:



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 .



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

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

$\mathcal{S} \rightarrow \bullet$: There are four candidate relations from the Sierpiński space to the singleton, but as we see in Example 4.2.7, not all of them are topological.

NOW WE NEED SOME ABSTRACTION. We cannot study the continuous relations between the singleton and the unit square case by case. We discover that continuous relations out of the singleton indicate arbitrary subsets, and that continuous relations into the singleton indicate arbitrary opens.

$\bullet \rightarrow \blacksquare$: Proposition 4.2.9 tells us that there are as many continuous relations from the singleton to the unit square as there are subsets of the unit square.

$\blacksquare \rightarrow \bullet$: Proposition 4.2.10 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 $\mathcal{S} \rightarrow \mathcal{S}$ TO CHECK. A case-by-case approach won't scale, so we could instead identify the building blocks of continuous relations with the same source and target space.

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

Proposition 4.2.13. If $R, S : X^\tau \rightarrow Y^\sigma$ are continuous relations, so are $R \cap S$ and $R \cup S$.

Reminder 4.2.5 (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}_τ and \mathfrak{b}_σ are bases for τ and σ respectively.

Reminder 4.2.6 (Product of relations). For relations between sets $R : X \rightarrow Y, S : A \rightarrow B$, the product relation $R \times S : X \times A \rightarrow Y \times B$ is defined to be

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

Example 4.2.7 (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 $\{0\}$.

Terminology 4.2.8. Call a continuous relation $\bullet \rightarrow X^\tau$ a state of X^τ , and a continuous relation $X^\tau \rightarrow \bullet$ a test of X^τ .

Proposition 4.2.9. States $R : \bullet \rightarrow X^\tau$ correspond with subsets of X .

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

Proposition 4.2.10. Tests $R : X^\tau \rightarrow \bullet$ 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 . \square

Reminder 4.2.11 (Union, intersection, and ordering of relations). Recall that relations $X \rightarrow Y$ 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.

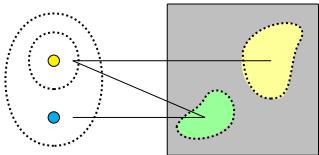


Figure 4.2: Regions of \blacksquare in the image of the yellow point alone will be coloured yellow, and regions in the image of both yellow and cyan will be coloured green:

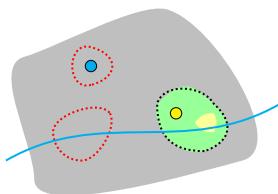


Figure 4.3: Regions in the image of the cyan point alone cannot be open sets by continuity, so they are either points or lines. Points and lines in cyan must be surrounded by an open region in either yellow or green, or else we violate continuity (open sets in red).

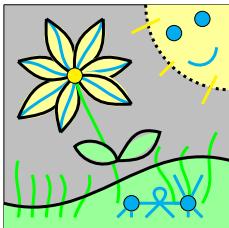


Figure 4.4: A continuous relation $\mathcal{S} \rightarrow \blacksquare$: "Flower and critter in a sunny field".

Proof. Replace \square with either \cup or \cap . For any non- \emptyset 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. \square

Corollary 4.2.14. Continuous relations $X^\tau \rightarrow Y^\sigma$ are closed under arbitrary union and finite intersection.

Hence, continuous relations $X^\tau \rightarrow Y^\sigma$ form a topological space where each continuous relation is an open set on the base space $X \times Y$, where the full relation $X \rightarrow Y$ is "everything", and the empty relation is "nothing".

A TOPOLOGICAL BASIS FOR SPACES OF CONTINUOUS RELATIONS

Reminder 4.2.15 (Topological Basis). $\mathfrak{b} \subseteq \tau$ is a basis of the topology τ if every $U \in \tau$ is expressible as a union of elements of \mathfrak{b} . Every topology has a basis (itself). Minimal bases are not necessarily unique.

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

Definition 4.2.16 (Partial Functions). A **partial function** $X \rightarrow Y$ 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 4.2.17 (Partial functions are a \cap -ideal). The intersection $f \cap R$ of a partial function $f : X \rightarrow Y$ with any other relation $R : X \rightarrow Y$ 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. \square

Lemma 4.2.18 (Any single edge can be extended to a continuous partial function). Given any $(x, y) \in X \times Y$, there exists a continuous partial function $X^\tau \rightarrow Y^\sigma$ 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) . \square

Proposition 4.2.19. Continuous partial functions form a topological basis for the space $(X \times Y)^{(\tau \rightarrow \sigma)}$, where the opens are continuous relations $X^\tau \rightarrow Y^\sigma$.

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

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

The \supseteq direction is evident, while the \subseteq direction follows from Lemma 4.2.18. By Lemma 4.2.17, every $R \cap F$ term is a partial function, and by Corollary 4.2.14, continuous. \square

$\mathcal{S} \rightarrow \mathcal{S}$: We can use Proposition 4.2.19 to write out the topological basis of continuous partial functions, from which we can take unions to find all the continuous relations, which we depict in Figure 4.6.

$\mathcal{S} \rightarrow \blacksquare$: Now we use the colour convention of the points in \mathcal{S} to "paint" continuous relations on the unit square "canvas", as in Figures 4.2 and 4.3. So each continuous relation is a painting, and we can characterise the paintings that correspond to continuous relations $\mathcal{S} \rightarrow \blacksquare$ 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.

$\blacksquare \rightarrow \mathcal{S}$: 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.

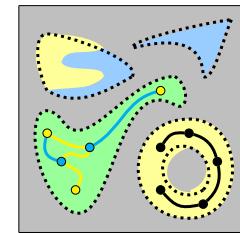


Figure 4.5: A continuous relation $\blacksquare \rightarrow \mathcal{S}$: "still math?". Black lines and dots indicate gaps.

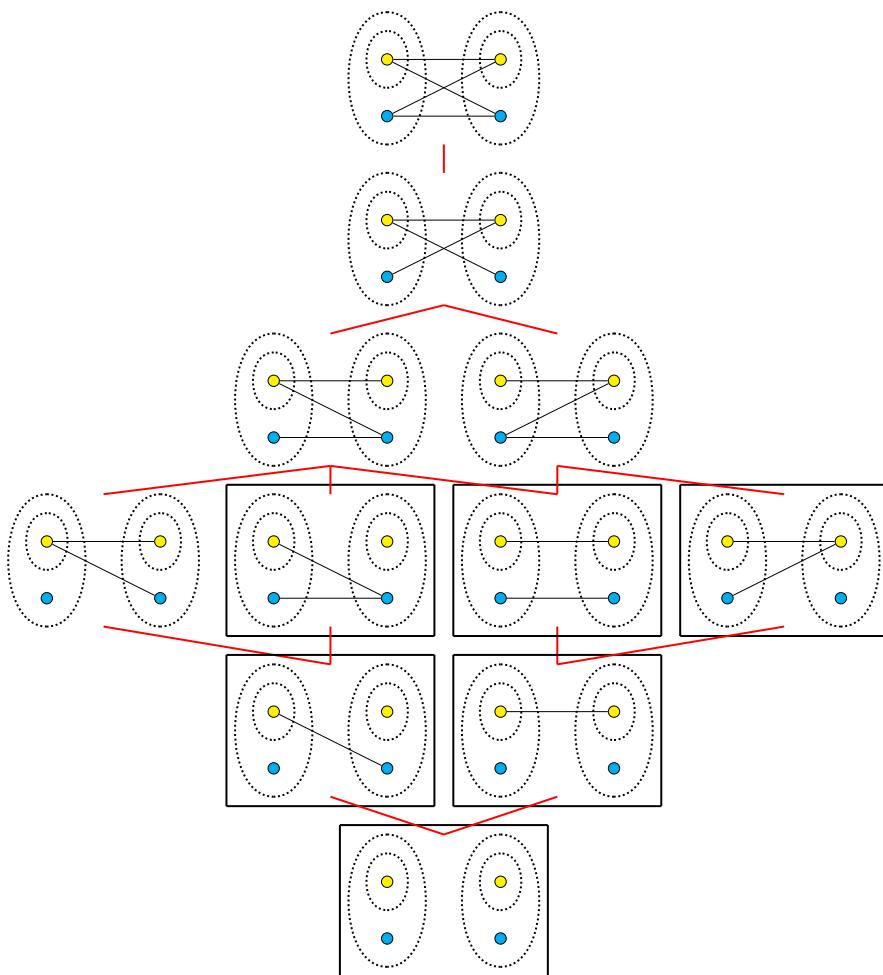


Figure 4.6: Hasse diagram of all continuous relations from the Sierpiński space to itself. Each relation is depicted left to right, and inclusion order is bottom-to-top. Relations that form the topological basis are boxed.

ONE MORE EXAMPLE FOR FUN, $[0, 1] \rightarrow \blacksquare$: 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 $[0, 1] \rightarrow \blacksquare$. By colour-coding $[0, 1]$ and controlling brushstrokes, we can do quite a lot. So, like it or not, here's a continuous relation



Now we would like to develop the abstract machinery required to *formally* paint pictures with words.



Figure 4.7: continuous functions $[0, 1] \rightarrow \blacksquare$ follow the naïve notion of continuity: a line one can draw on paper without lifting the pen off the page.



Figure 4.8: An example of a continuous relation is "(countably) many (open-ended) lines, each of which one can draw on paper without lifting the pen off the page."

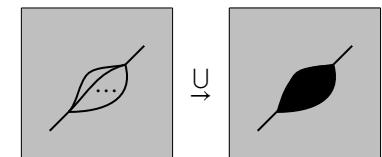


Figure 4.9: We can control the thickness of the brushstroke, by taking the union of (uncountably) many lines.



Figure 4.10: 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.

Proposition 4.3.1. Continuous functions are always continuous. If $f : X^\tau \rightarrow Y^\sigma$ 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. \square

Corollary 4.3.2. There is a faithful, identity-on-objects monoidal embedding $\mathbf{Top} \hookrightarrow \mathbf{ContRel}$.

Proposition 4.3.3. The identity relation $X \rightarrow X$ relates anything to itself. It is defined $\{(x, x) : x \in X\} \subseteq X \times X$. The identity relation is always continuous.

Proof. The preimage of any open set under the identity relation is itself, which is open by assumption. The identity relation is also the trivial continuous function from a space to itself, so this also follows from Proposition 4.3.1. \square

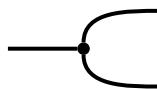


Figure 4.11: The copy map $X^\tau \rightarrow X^\tau \times X^\tau$, $\{(x, \begin{pmatrix} x \\ x \end{pmatrix}) \mid x \in X\}$.

Proposition 4.3.4. Copy maps are continuous relations.

Proof. For a direct proof, we draw on the fact that given a basis \mathfrak{b} for a topology τ , ordered pairs of \mathfrak{b} form a basis for the product topology $\tau \times \tau$. To show that the preimage of an open in $\tau \times \tau$ is open in τ , we may consider the preimages under the copy map of basis elements of $\tau \times \tau$, which are intersections of pairs of basis elements of τ , and hence definitionally open. By closure of opens under arbitrary unions, all opens of $\tau \times \tau$ have an open preimage in τ . \square

4.3 The category $\mathbf{ContRel}$

Definition 4.3.9 ($\mathbf{ContRel}$). The (purported) category $\mathbf{ContRel}$ has topological spaces for objects and continuous relations for morphisms.

Proposition 4.3.10 ($\mathbf{ContRel}$ is a category). continuous relations form a category $\mathbf{ContRel}$.

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 \mathbf{Rel} . \square

4.3.1 Symmetric Monoidal structure

Proposition 4.3.11. $(\mathbf{ContRel}, \bullet, X^\tau \otimes Y^\sigma := (X \times Y)^{(\tau \times \sigma)})$ is a symmetric monoidal category.

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

TENSOR PRODUCT: For objects, $X^\tau \otimes Y^\sigma$ has base set $X \times Y$ equipped with the product topology $\tau \times \sigma$. For morphisms, $R \otimes S$ the product of relations. We show that the tensor of continuous relations is again a continuous relation. Take continuous relations $R : X^\tau \rightarrow Y^\sigma, S : A^\alpha \rightarrow B^\beta$, 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}_σ and \mathfrak{b}_β respectively. Since for any relations we have that $R(A \cup B) = R(A) \cup R(B)$ and $(R \times S)^\dagger = R^\dagger \times S^\dagger$:

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

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

THE NATURAL ISOMORPHISMS ARE INHERITED FROM **Rel**. We will be explicit with the unit, 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 $\lambda_{X^\tau} : \bullet \times X^\tau \rightarrow X^\tau := \{(\binom{\star}{x}, x) \mid x \in X\}$, and we reverse the pairs to obtain the inverse $\lambda_{X^\tau}^{-1}$. 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 $\alpha_{X^\tau Y^\sigma Z^\rho} : ((X \times Y) \times Z)^{((\tau \times \sigma) \times \rho)} \rightarrow (X \times (Y \times Z))^{(\tau \times (\sigma \times \rho))}$ are inherited from **Rel**. They are:

$$\alpha_{X^\tau Y^\sigma Z^\rho} := \{((\binom{x}{y}, z), (x, \binom{y}{z})) \mid x \in X, y \in Y, z \in Z\}$$

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 $\theta_{X^\tau Y^\sigma} : (X \times Y)^{\tau \times \sigma} \rightarrow (Y \times X)^{\sigma \times \tau}$ are defined:

$$\{(\binom{x}{y}, \binom{y}{x}) \mid x \in X, y \in Y\}$$

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.

4.3.2 Rig category structure

Definition 4.3.12 (Biproducts and zero objects). A *biproduct* is simultaneously a categorical product and coproduct. A *zero object* is both an initial and a terminal object. **Rel** has biproducts (the coproduct of sets equipped with reversible injections) and a zero object (the empty set).

Proposition 4.3.13. **ContRel** has a zero object.

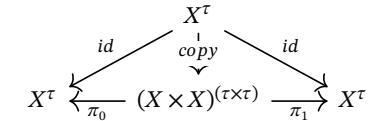


Figure 4.12: An alternative proof of Proposition 4.3.4 follows from Proposition 4.3.1, Corollary 4.3.2, and the definition of the product topology as the coarsest topology that satisfies categorical product for the diagram above.



Figure 4.13: The *everything* state is the relation $\{(\star, x) \mid x \in X\}$, notated as above.

Proposition 4.3.5. The everything states are continuous relations.

Proof. The preimage of any subset of X – in particular the opens – is the whole of the singleton space, which is open. \square



Figure 4.14: The *delete test*, $\{(x, \star) \mid x \in X\}$.

Proposition 4.3.6. The delete tests are continuous relations.

Proof. There are only two opens in the singleton space. The preimage of the empty set is the empty set, and the preimage of the singleton is the whole of X ; both are opens in X by definition. \square

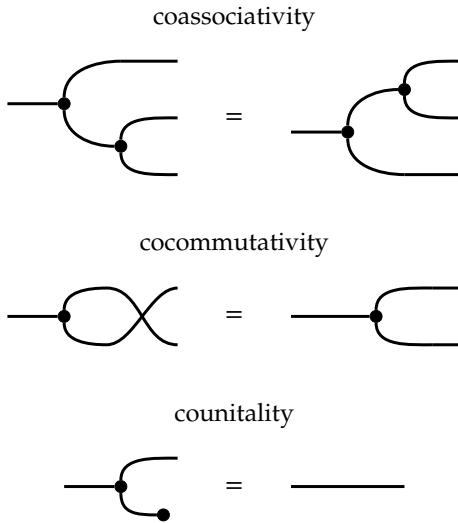


Figure 4.15: Copy and delete satisfy the above properties, expressed as diagrammatic equations.

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

Proposition 4.3.14. **ContRel** has biproducts.

Proof. The biproduct of topologies X^τ and Y^σ is their direct sum topology $(X \sqcup Y)^{(\tau+\sigma)} - \tau \sqcup \sigma$. 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 $R : X^\tau \rightarrow Z^\rho$ and $S : Y^\sigma \rightarrow Z^\rho$, 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. \square

Remark 4.3.15. Biproducts yield another symmetric monoidal structure which the \times monoidal product distributes over appropriately to yield a rig category. Throughout the chapter we will use \sqcup , but we could have also "diagrammatised" \sqcup by treating it as a monoid internal to **ContRel** viewed as a symmetric monoidal category with respect to the biproduct. There are at least two diagrammatic formalisms for rig categories that we could have used such as [CDH20], but it gets too visually complicated, especially when we sometimes take unions over arbitrary indexing sets, which is alright in topology but not depictable as a finite diagram in the \oplus -structure. A neat fact that follows is that a topological space is compact precisely when any arbitrarily indexed \sqcup 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.

4.3.3 Monoidal (co!)closure

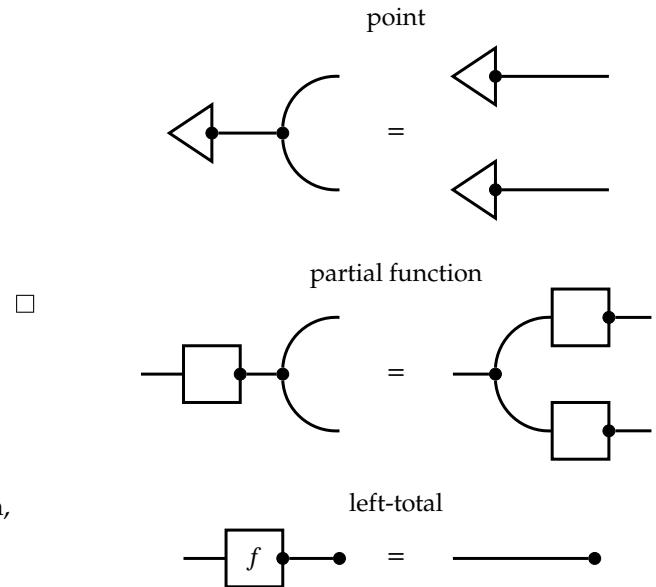
Definition 4.3.16 (Closure type). Recall by Proposition 4.2.13 that continuous relations $X^\tau \rightarrow Y^\sigma$ form a topological space. Denote this space $(X \times Y)^{(\tau \multimap \sigma)}$

The permissible continuous relations $X^\tau \rightarrow Y^\sigma$ are tests on $(X \times Y)^{(\tau \multimap \sigma)}$. **ContRel** is not monoidal closed, because taking a closure type as an input to the evaluator would permit arbitrary subsets of $X \times Y$ as arguments. So what we seek instead is a coevaluation, where the closure type is an output. This is not as straightforward as it is in strongly compact closed **Rel**, where we may use cups and caps for process-state duality (CJ-isomorphism), because in **ContRel** we have cups *but no caps*. However, we may exploit the fact that discrete topologies behave like plain sets (see Lemma 4.3.23) and the observation that we may coarsen discrete topologies into target topologies, which is essentially enough to recover monoidal coclosed structure.

Proposition 4.3.17. For any X^τ and Y^σ , $\tau \times \sigma \subseteq \tau \multimap \sigma$; the product topology is coarser than the corresponding closure topology.

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

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



Example 4.3.18 ($\tau \multimap \sigma \not\subseteq \tau \times \sigma$). Recalling Proposition 4.2.9, let $\tau = \{\emptyset, \{\star\}\}$ on the singleton, and σ be an arbitrary nondiscrete topology on base space Y . $(\{\star\} \times Y)^{(\tau \times \sigma)}$ is isomorphic to Y^σ , but $(\{\star\} \times Y)^{(\tau \multimap \sigma)}$ is isomorphic to the discrete topology Y^* . For a more concrete example, consider the Sierpiński space \mathcal{S} again, along with the topological relation $\{(0, 0), (1, 0), (1, 1)\} \subset \mathcal{S} \times \mathcal{S}$; due to the presence of $(0, 0)$, this topological relation cannot be formed by a union of basis elements of the product topology, which are:

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

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

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

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

Definition 4.3.19 (Coarsening). Where $\tau \supseteq \rho$ are topologies on X , the identity-on-elements relation $X^\tau \rightarrow X^\rho$ is continuous; the relational converse of the identity is the identity, which witnesses opens of ρ as opens of τ , but the converse is not unless $\tau = \rho$. We denote these *coarsening* relations as:

$$X^\tau \quad \text{---} \circ \text{---} \quad X^\rho$$

Definition 4.3.20 (Pseudo-compare). For any X^τ , the relation $X^* \times X^\tau \rightarrow X^\tau$ defined on objects as

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

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

Figure 4.16: Relations that interact with copy and delete are nice, and we notate them with the same black dots as for copy and delete to mark them. States are singletons, or points, when they are copyable (and non-empty). Partial continuous functions are those that commute with copy. Left-total relations are those that commute with delete. Continuous functions are those that satisfy the latter two criteria.

Proposition 4.3.7. The **full relation** $X \rightarrow Y$ relates everything to everything. It is all of $X \times Y$. Full relations are always continuous.

Proof. For a direct proof, the preimage of any subset of Y under the full relation is the whole of X , which is open by definition. Alternatively, the full relation is the composite of delete and then everything. \square



Figure 4.17: The **empty state** $\bullet \rightarrow X^\tau$ and **empty test** relate nothing. The **empty relation** $X^\tau \rightarrow Y^\sigma$ is the composite of empty tests and states, and relates nothing: as a relation it is $\emptyset \subset X \times Y$.

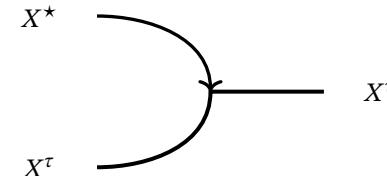
Proposition 4.3.8. Empty states, tests, and relations are continuous.

Proof. The preimage of any empty relation is the empty set, which is definitionally open. \square

$$\begin{array}{l}
 \text{(b) one} \\
 \bullet - \bullet = \boxed{\quad} \\
 \text{zero} \\
 \bullet - \bullet = \bullet - \bullet \\
 // \qquad // \\
 \bullet - \bullet = \boxed{\quad} \\
 // \\
 \bullet \qquad \bullet
 \end{array}$$

Figure 4.18: There are two scalars: the unit, and the zero scalar. Both the one and zero scalars are idempotent, which diagrammatically means that we may freely make and merge copies of them.

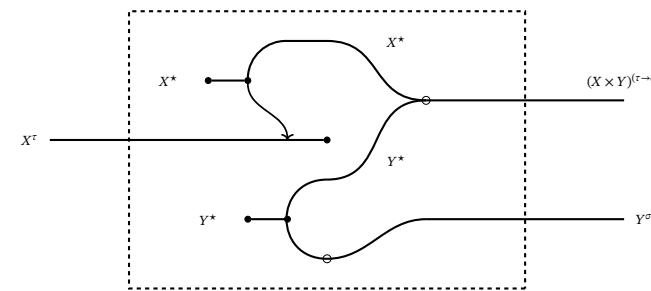
which are opens in $X^* \times X^\tau$. We denote these *pseudo-compare* maps:



Proposition 4.3.21. Where \star_X and \star_Y are discrete topologies on X and Y , $\star_X \times \star_Y = \star_X \multimap \star_Y$.

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

Proposition 4.3.22 (ContRel is monoidal coclosed). The coevaluation map is:



Proof. The string diagram itself demonstrates that the coevaluation is continuous, since we have demonstrated that **ContRel** is symmetric monoidal, and we know that copy (Prop. 4.3.4), everything (Prop. 4.3.5), coarsenings (Defn. 4.3.19) and pseudo-compares (Defn. 4.3.20) are continuous. What remains to be demonstrated is that the coevaluation behaves like one; i.e. that plugging in a continuous relation expressed as a test into the closure type of the coevaluator recovers that continuous relation. This can be shown diagrammatically by equating the underlying relations on sets in **Rel**.

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

Rel, and in particular, proposition 4.3.21 deals with the 2-1 coarsening. The expression inside the closure test is obtained by CJ-isomorphism using strong compact closure in **Rel**. The pseudo-compares in **ContRel** becomes an honest compare in **Rel**. The second equation then follows by frobenius. \square

THAT'S ALL WE NEED FOR THE DIAGRAMS. The remainder of this section are endnotes for category theorists addressing the question of how **ContRel** relates to **Rel** and **Top**, and some conceptual motivations for topological relations. If none of that interests you, ignore the main body: the margins carry on with diagrammatic facts about **ContRel**.

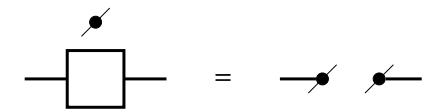


Figure 4.19: There is a zero-morphism for every input-output pair of objects in **ContRel**, which is diagrammatically the composition of the empty test and state. Zero scalars turn any relation into a zero relation. Substituting the zero relation into the LHS of the above equation means that zero relations also spawn zero scalars.

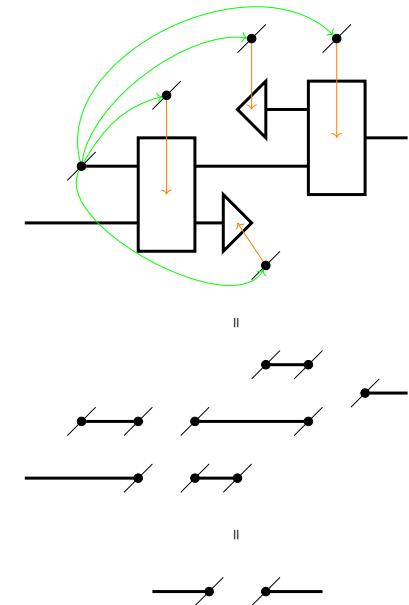


Figure 4.20: So, whenever a zero-process appears in a diagram, it spawns zero scalars which infect all other processes, turning them all into zero-processes. The same holds for whenever a zero-scalar appears; it makes copies of itself to infect all other processes.

4.3.4 Category-theoretic endnotes

ContRel AND **Rel** ARE RELATED BY A FREE-FORGETFUL ADJUNCTION

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

Lemma 4.3.23 (Any relation R between discrete topologies is continuous). *Proof.* All subsets in a discrete topologies are open. \square

Definition 4.3.24 ($L: \mathbf{Rel} \rightarrow \mathbf{ContRel}$). We define the action of the functor L :

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

On morphisms $L(X \xrightarrow{R} Y) := X^* \xrightarrow{R} Y^*$, the existence of which in **ContRel** is provided by Lemma 4.3.23.

Evidently identities and associativity of composition are preserved.

Definition 4.3.25 ($R: \mathbf{ContRel} \rightarrow \mathbf{Rel}$). 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 4.3.26 ($RL = 1_{\mathbf{Rel}}$). The composite RL (first L , then R) is precisely equal to the identity functor on **Rel**.

Proof. On objects, $FU(X) = F(X^*) = X$. On morphisms, $FU(X \xrightarrow{R} Y) = F(X^* \xrightarrow{R} Y^*) = X \xrightarrow{R} Y$ \square

Reminder 4.3.27 (Coarser and finer). Given a set of points X with two topologies X^τ and X^σ , if $\tau \subset \sigma$, we say that τ is coarser than σ , or σ is finer than τ .

Lemma 4.3.28 (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, $U \in \tau$. \square

Proposition 4.3.29 ($L \dashv R$). *Proof.* We verify the triangular identities governing the unit and counit of the adjunction, which we first provide. By Lemma 4.3.26, we take the natural transformation $1_{\mathbf{Rel}} \Rightarrow RL$ 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 $\mathbf{ContRel}$ is granted by Lemma 4.3.28.

$$\epsilon_{X^\tau} : X^* \rightarrow X^\tau := \{(x, x) : x \in X\}$$

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

Now we evaluate $R \xrightarrow{\eta R} RLR \xrightarrow{R\epsilon} R$ at an arbitrary object (topological space) $X^\tau \in \mathbf{ContRel}$. $R(X^\tau) = X = RLR(X^\tau)$, where the 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, $R(\epsilon_{X^\tau}) = R(X^* \xrightarrow{1_X} X^\tau) = X \xrightarrow{1_X} X = 1_X$. So $\eta_R; R\epsilon = R$, as required. \square

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

Reminder 4.3.30 (The category \mathbf{Loc}). [nLab] 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* $\phi : A \rightarrow B$ is a function between frames that preserves finite meets and arbitrary joins. The category \mathbf{Frm} has frames as objects and frame homomorphisms as morphisms. The category \mathbf{Loc} is defined to be \mathbf{Frm}^{op} .

Remark 4.3.31. Here are informal intuitions to ease the definition. The lattice of open sets of a given topology ordered by inclusion forms a frame – observe the analogy "arbitrary unions" : "all joins" :: "finite intersections" : "finite meets". Closure under arbitrary joins guarantees a maximal element corresponding to the open set that is the whole space. So frames are a setting to speak of topological structure alone, without referring to a set of underlying points, hence, pointless topology. Observe that in the definition of continuous functions, open sets in the *codomain* must correspond (uniquely) to open sets in the *domain* – so every continuous function induces a frame homomorphism going in the opposite direction that the function does between spaces, hence, to obtain the category \mathbf{Loc} such that directions align, we reverse the arrows of \mathbf{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.

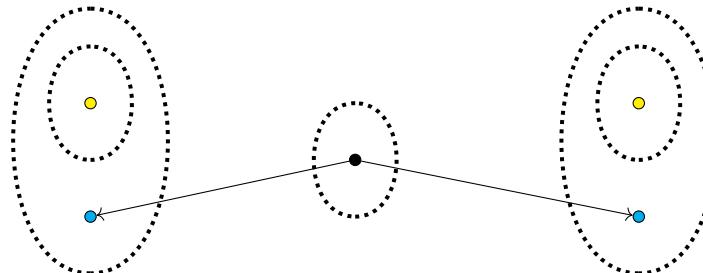
Definition 4.3.32 ($U : \mathbf{ContRel} \rightarrow \mathbf{Loc}$). On objects, U sends a topology X^τ to the frame of opens in τ , which we denote $\hat{\tau}$.

On morphisms $R : X^\tau \rightarrow Y^\sigma$, the corresponding partial frame morphism $\hat{\tau} \leftarrow \hat{\sigma}$ (notice the direction reversal for \mathbf{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 4.3.33 (U has no left adjoint). *Proof.* Seeking contradiction, if U were a right adjoint, it would preserve limits. The terminal object in \mathbf{Loc} is the two-element lattice $\perp < \top$, where the unique frame homomorphism to any \mathcal{L} sends \top to the top element of \mathcal{L} and \perp to the bottom element. In $\mathbf{ContRel}$, the empty topology $\mathbf{0} = (\emptyset, \{\emptyset\})$ is terminal (and initial). However, $U\mathbf{0}$ is the singleton lattice, not $\perp < \top$ (which is the image under U of the singleton topology). \square

This is a rather frustrating result, because U does turn continuous relations into backwards frame homomorphisms on lattices of opens; see Proposition 4.2.13, and note that in the frame of opens associated with a topology, the empty set becomes the bottom element. The obstacle is the fact that the empty topology is both initial and terminal in $\mathbf{ContRel}$. We may be tempted to try treating U as a right adjoint going to \mathbf{Frm} instead, but then the monad induced by the injunction on \mathbf{Loc} would trivialise: left adjoints preserve colimits, so any putative left adjoint F must send $\perp < \top$ (initial in \mathbf{Frm} by duality) to the empty topology, and the empty topology as terminal object must be sent to the terminal singleton frame, which implies that the monad UF on \mathbf{Frm} sends everything to the singleton lattice.

WHY NOT $\text{SPAN}(\mathbf{Top})$? 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 \mathbf{Top} that corresponds to a relation that doesn't live in $\mathbf{ContRel}$. It is the span with the singleton as apex, with maps from the singleton to the closed points of a two Sierpiński spaces.



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

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

This condition would be nice to have because it witnesses $K_{\mathcal{P}}$ as precisely K_T restricted to the discrete topologies, so that T really behaves as a conservative generalisation of the notion of relations to accommodate topologies. As a consequence of this condition, we may observe that discrete topologies X^* must be sent to discrete topologies on their powerset $\mathcal{P}X^*$. In particular, this means the singleton topology is sent to the discrete topology on a two-element set; $T\bullet = 2$. This sinks us. We know from Proposition 4.2.10 that the continuous relations $X^\tau \rightarrow \bullet$ are precisely the open sets of τ , which correspond to continuous functions into Sierpiński space $X^\tau \rightarrow \mathbb{S}$, and $\mathbb{S} \neq 2$.

WHERE IS THE TOPOLOGY COMING FROM?

It is category-theoretically natural to ask whether $\mathbf{ContRel}$ 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 $\mathbf{ContRel}$ in the literature. However, we do have a free-forgetful adjunction between $\mathbf{ContRel}$ and \mathbf{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 considering it a preorder will suffice.) The fibre over a relation $R : A \rightarrow B$ 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 $\mathbf{Rel} \rightarrow \mathcal{C}$, where \mathcal{C} is a category with preorders for objects and boolean-enriched profunctors as morphisms, and the functor encodes topological data by sending sets in **Rel** to preorders of all possible topologies, and relations to profunctors. I have deliberately left this as a sketch because it doesn't seem worth it to view something so simple in such a complex way (I accept the charges of hypocrisy having just used weak n -categories to present TAGs.)

4.4 Populating space with shapes using sticky spiders

In this section, we seek to process-theoretically characterise disjoint collections of open sets of a space, so that we can play with doodles on the page as formal objects. It turns out that in **ContRel**, we can express them as idempotents that interact with spiders in a certain way.

Example 4.4.1 (The copy-compare spiders of **Rel** are not always continuous). The compare map for the Sierpiński space is not continuous, because the preimage of $\{0, 1\}$ is $\{(0, 0), (1, 1)\}$, which is not open in the product space of \mathcal{S} with itself.

Reminder 4.4.2 (copy-compare spiders of **Rel**). For a set X , the *copy* map $X \rightarrow X \times X$ is defined:

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

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

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

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

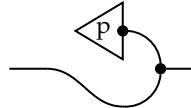
$$\{(x, \star) : x \in X\}$$

and *everything*:

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

Proposition 4.4.3. The copy map is part of a special commutative frobenius algebra iff the topology is discrete

Proof. Discrete topologies inherit the usual copy-compare spiders from \mathbf{Rel} , so we have to show that when the copy map is part of 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 tests whether the input is the singleton $\{p\}$ – when all singletons are open, the topology is discrete. As a lemma, we show that comparing distinct points $p \neq q$ yields the empty state.

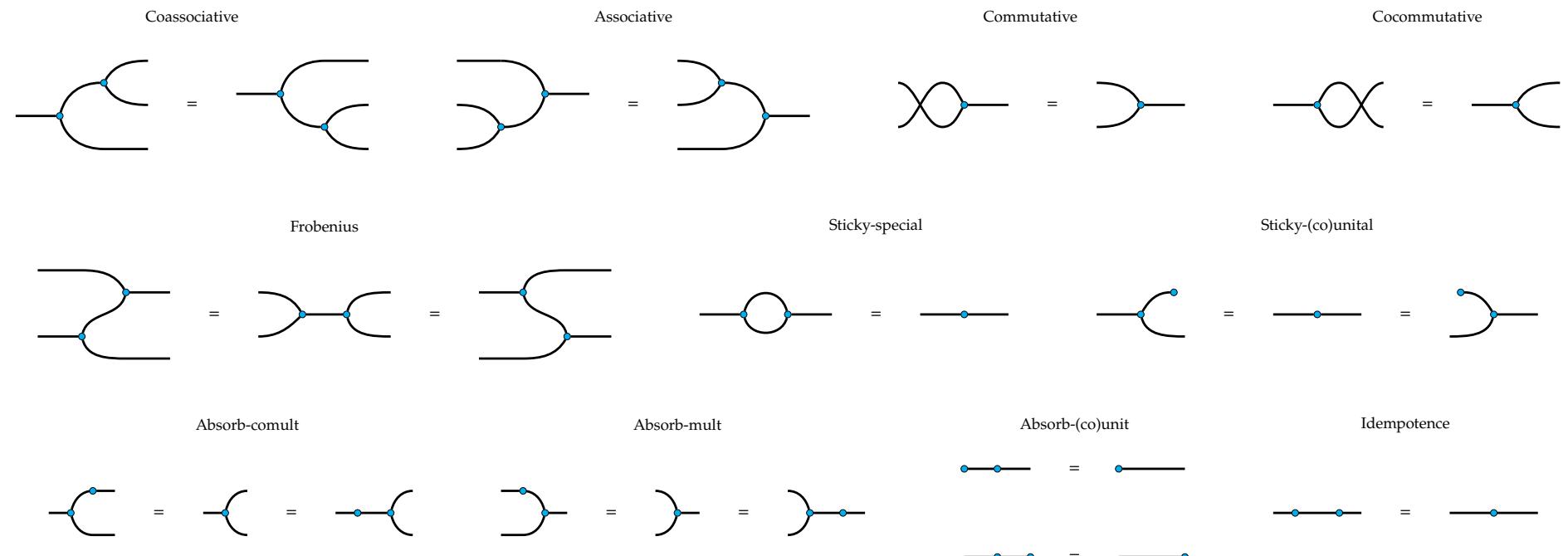
The (zero) implication follows since $p \neq q$ by assumption, so we know that deleting the comparison of p and q cannot be the unit scalar, and so must be the zero scalar, hence the comparison of p and q is the empty state. Now, the following case analysis shows that our open set only contains the point p .

$$\begin{array}{c}
 \text{if } p \neq q \\
 \text{if } p = q
 \end{array}
 \quad
 \begin{array}{c}
 \text{= } \\
 \text{= } \\
 \text{= }
 \end{array}
 \quad
 \begin{array}{c}
 \text{= } \\
 \text{= } \\
 \text{= }
 \end{array}$$

1

Definition 4.4.4 (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 is replaced by the idempotent e .

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.



WE CAN USE SPLIT IDEMPOTENTS TO TRANSFORM COPY-SPIDERS FROM DISCRETE TOPOLOGIES TO STICKY-SPIDERS ON OTHER SPACES.

Reminder 4.4.5 (Split idempotents). An **idempotent** in a category is a map $e : A \rightarrow A$ such that

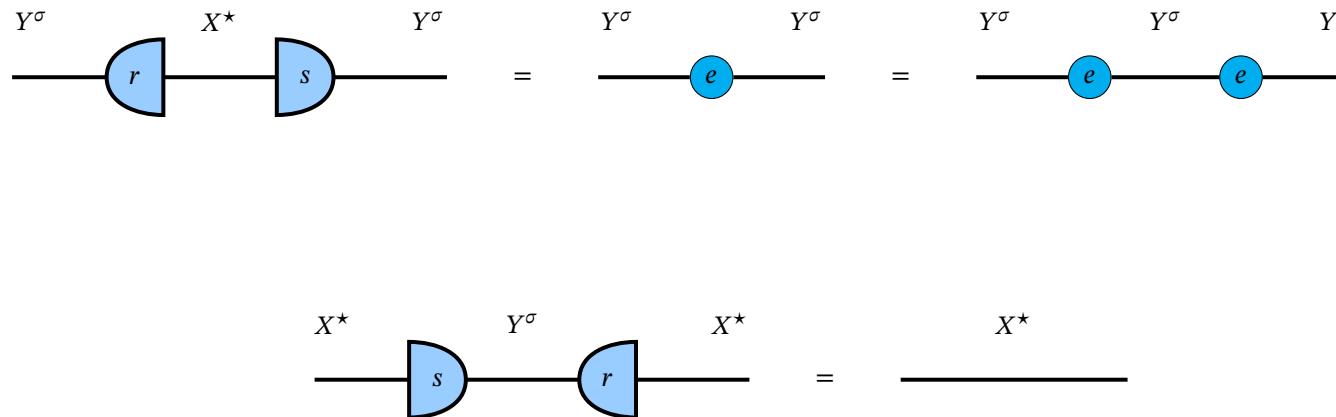
$$A \xrightarrow{e} A \xrightarrow{e} A = A \xrightarrow{e} A$$

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

$$A \xrightarrow{e} A = A \xrightarrow{r} B \xrightarrow{s} A$$

$$B \xrightarrow{s} A \xrightarrow{r} B = B \xrightarrow{id} B$$

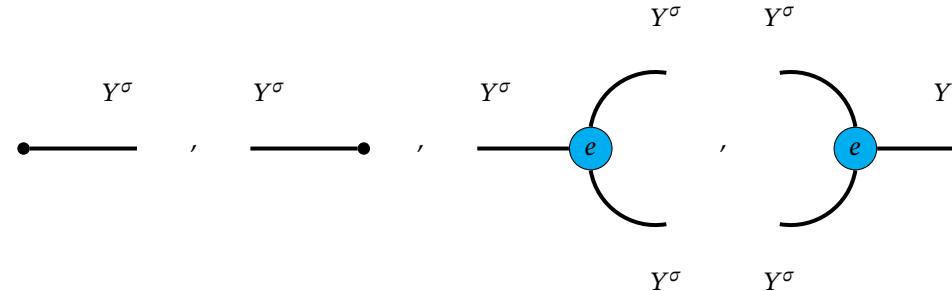
We can graphically express the behaviour of a split idempotent e as follows, where the semicircles for the section and retract r, s form a visual pun. Recall that X^* denotes the discrete topology on the set X .



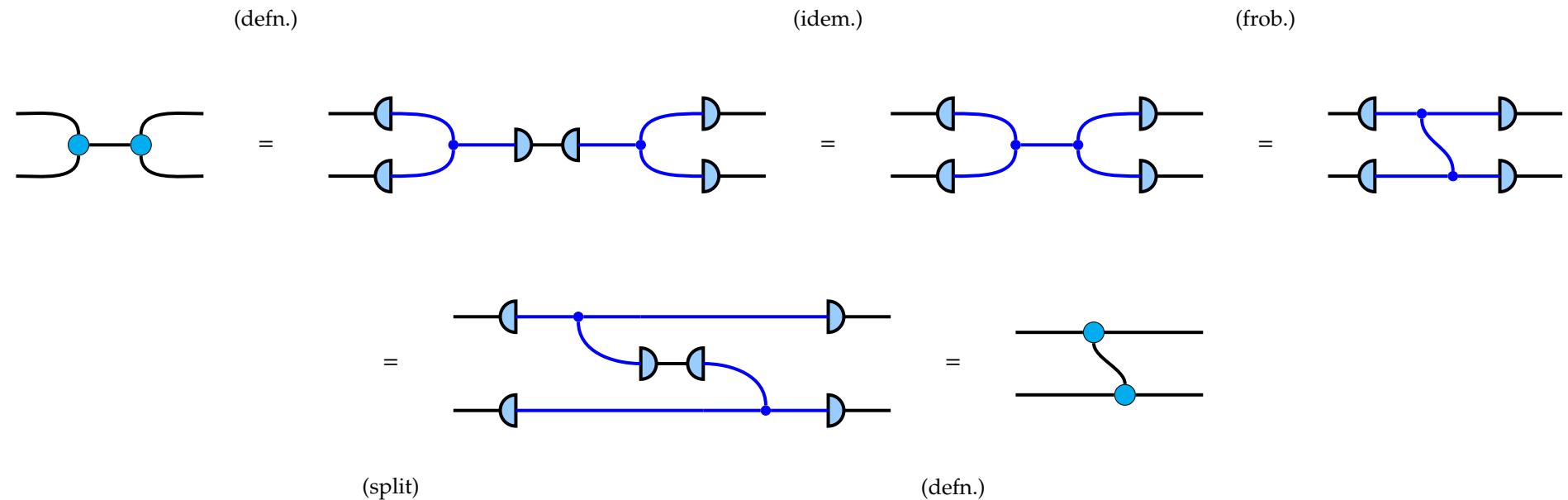
Construction 4.4.6 (Sticky spiders from split idempotents). Given an idempotent $e : Y^\sigma \rightarrow Y^\sigma$ that splits through a discrete topology X^* , we construct a new (co)multiplication as follows:



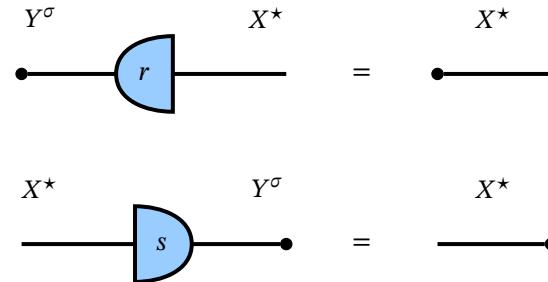
Proposition 4.4.7 (Every idempotent that splits through a discrete topology gives a sticky spider). The following is a sticky spider:



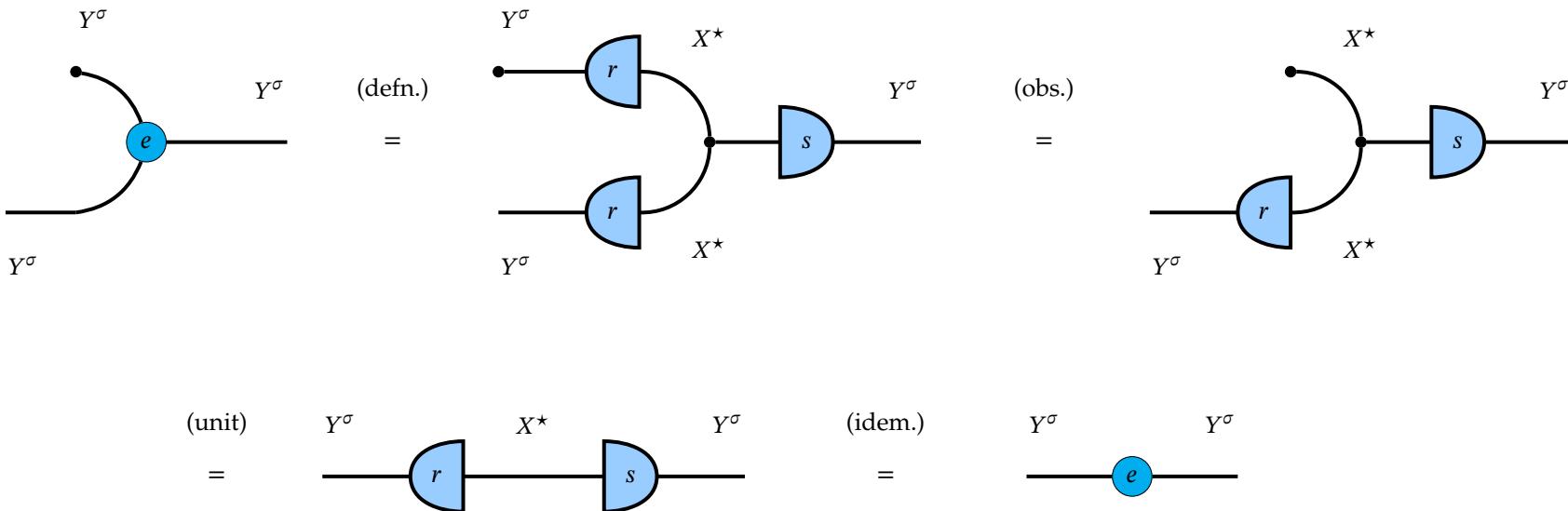
We can check that Construction 4.4.6 satisfies the frobenius rules as follows. We only present one equality; the rest follow the same idea.



To verify the sticky spider rules, we first observe that since $X^* \xrightarrow{s} Y^\sigma \xrightarrow{r} X^* = X^* \xrightarrow{id} X^*$, r must have all of X^* in its image, and s must have all of X^* in its preimage, so we have the following:



Now we show that e-unitality holds:



The proofs of e-countability, and e-speciality follow similarly.

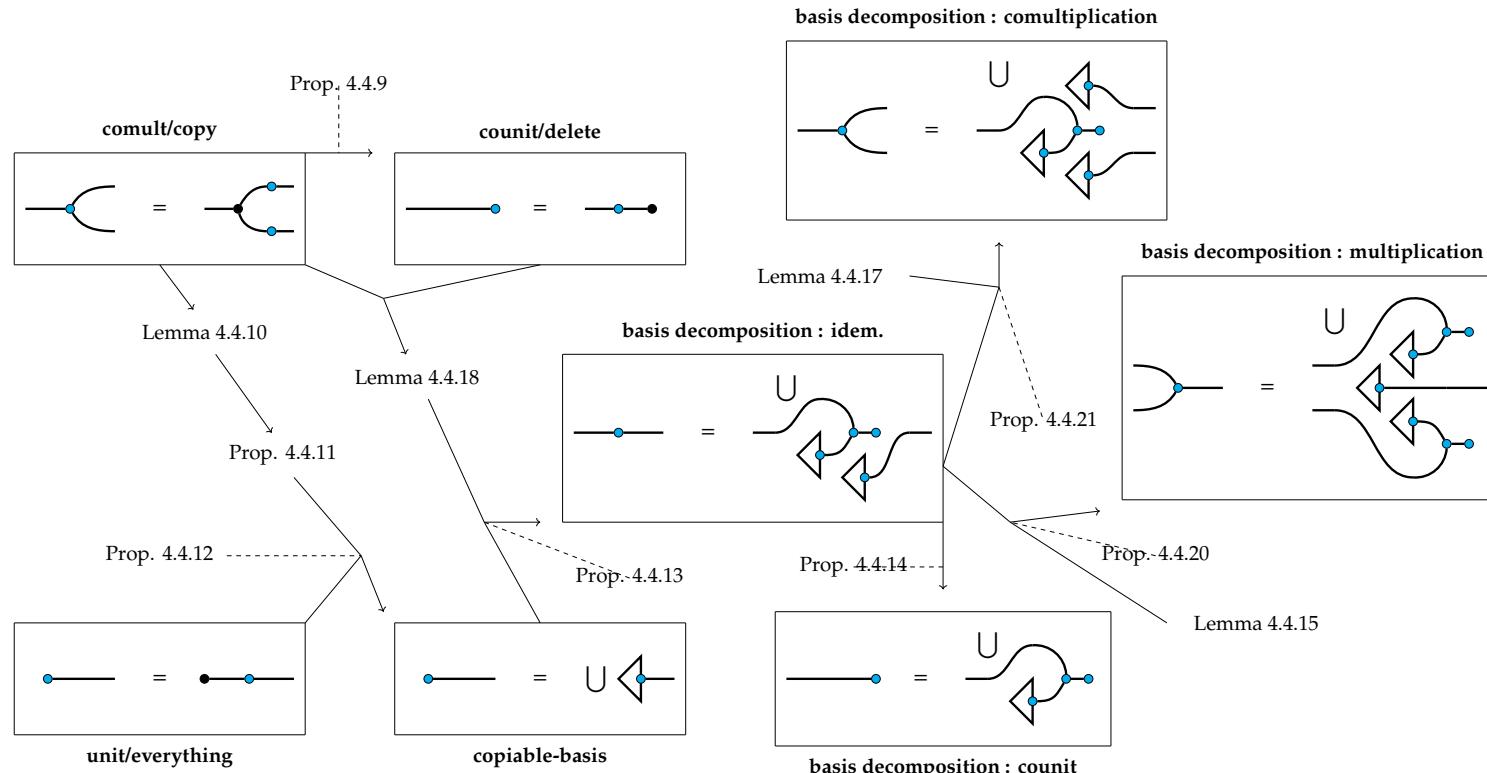
WE CAN PROVE A PARTIAL CONVERSE OF PROPOSITION 4.4.7: we can identify two diagrammatic equations that tell us precisely when a sticky spider has an idempotent that splits through some discrete topology.

Theorem 4.4.8. A sticky spider has an idempotent that splits through a discrete topology if and only if in addition to the sticky spider equalities, the following relations are also satisfied.

Unit/everything

Comult/copy

The proof is involved, so here is a map of lemmas and propositions.



Proposition 4.4.9 (comult/copy implies counit/delete).

$$\text{Diagram 1} = \text{Diagram 2} \Rightarrow \text{Diagram 3} = \text{Diagram 4}$$

Proof.

(comult/copy) (del)

$$\text{Diagram 1} = \text{Diagram 2} \subseteq \text{Diagram 3}$$

“ (del) (e-unit) // (copy-del)

$$\text{---} \circ \text{---} = \text{---} \circ \text{---}$$

So:

$$\text{---} \bullet = \text{---} \bullet \text{---} \bullet = \text{---} \bullet \text{---} \bullet = \text{---} \bullet \text{---} \text{---}$$

1

Lemma 4.4.10 (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 $e(\{y\}) = \emptyset$ or $e(\{x\}) = e(\{y\})$. Diagrammatically:

$$\begin{array}{c} \text{Diagram A} \\ \in \end{array} \quad \begin{array}{c} \text{Diagram B} \\ \Rightarrow \end{array} \quad \begin{array}{c} \text{Diagram C} \\ = \end{array} \quad \text{Or} \quad \begin{array}{c} \text{Diagram D} \end{array}$$

Suppose

$$\begin{array}{c} \text{Diagram A} \\ \text{Diagram B} \end{array} \neq \begin{array}{c} \text{Diagram C} \end{array}$$

For the claim, we seek:

$$\begin{array}{c} \text{Diagram A} \\ = \\ \text{Diagram B} \end{array}$$

We have the following inclusion:

The diagram illustrates a sequence of five steps in a derivation:

- (premise)**: A diagram showing two parallel arrows pointing left, each with a dot at its tail and a blue dot at its head.
- (absorb-mult)**: A diagram showing two parallel arrows pointing left, with a red curved arrow labeled with a red \sqcup symbol above them. This step is followed by an equals sign.
- (copy)**: A diagram showing two parallel arrows pointing left, with the bottom arrow having a blue dot at its head and a black dot at its tail.
- (comult/copy)**: A diagram showing two parallel arrows pointing left, with the bottom arrow having a black dot at its head and a blue dot at its tail.
- (e-special)**: A diagram showing two parallel arrows pointing left, with both arrows having black dots at their heads and tails.

(assumption)

Since:  \neq  and

we have that:

So we have the following **equality**:

$$\begin{array}{c}
 \text{Diagram 1:} \\
 \text{(frob.)} \quad \text{||} \quad \text{(comult/copy)} \quad = \quad \text{(e-spider)} \quad = \quad \text{(e-copy)} \quad \text{||} \\
 \text{Diagram 2:} \\
 \text{||} \quad \text{||} \quad = \quad \text{(e-spider)} \quad = \quad \text{(e-copy)} \quad = \quad \text{Diagram 3:}
 \end{array}$$

$$\text{Diagram 4:} \quad = \quad \text{Diagram 5:} \quad = \quad \text{Diagram 6:}$$

Which implies:

$$\text{Diagram 7:} \quad = \quad \text{Diagram 8:} \quad = \quad \text{Diagram 9:}$$

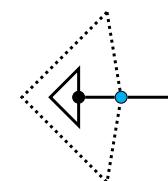
and symmetrically,

$$\text{Diagram 10:} \quad = \quad \text{Diagram 11:}$$

So we have the claim:

$$\text{Diagram 12:} \quad = \quad \text{Diagram 13:}$$

Proposition 4.4.11 (e of any point is e -copyable).



Proof.

$$\begin{array}{ccccccc}
 \text{(idem.)} & & & \text{(Lem. 4.4.10)} & \text{(point)} & & \text{(\cup \& comult/copy)} \\
 \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} & = & \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} & = & \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} & = & \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \\
 & & \cup & \cup & \cup & \cup & \cup \\
 & & \text{Diagram 3} & \text{Diagram 4} & \text{Diagram 5} & \text{Diagram 6} & \text{Diagram 7} \\
 & & \cup & \cup & \cup & \cup & \cup \\
 & & \text{Diagram 8} & \text{Diagram 9} & \text{Diagram 10} & \text{Diagram 11} & \text{Diagram 12}
 \end{array}$$

1

Proposition 4.4.12 (The unit is the union of all e -copiables).

$$\text{---} = \text{U} \triangleleft$$

Proof.

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

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

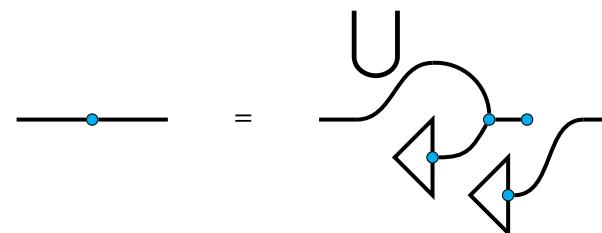
$$\begin{array}{c}
 \text{Diagram showing the equivalence of } U \text{ and } U^\dagger \text{ in the context of unitary evolution:} \\
 \text{Left: } U \text{ (unitary) and } U^\dagger \text{ (unitary).} \\
 \text{Middle: } U \text{ and } U^\dagger \text{ connected by a dashed line, representing } U^\dagger U = I. \\
 \text{Right: } U \text{ and } U^\dagger \text{ connected by a solid line, representing } U U^\dagger = I. \\
 \text{Bottom: Labels "(unit/evr.)" and "(Prop. 4.4.11)" indicating the properties shown.}
 \end{array}$$

So the containment must be an equality.

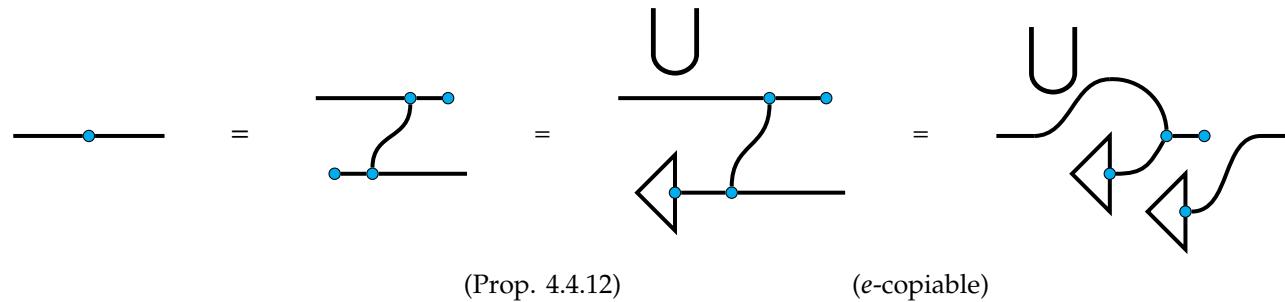
$$\text{---} = \text{U}^\wedge$$

1

Proposition 4.4.13 (e -copyable decomposition of e).



Proof.



□

Proposition 4.4.14 (e -copyable decomposition of counit).

$$\text{---} \bullet = \text{---} \text{---} \text{---} \text{---}$$

Proof.

(Prop. 4.4.13)

$$\text{---} \bullet = \text{---} \bullet \text{---} \bullet = \text{---} \text{---} \text{---} \text{---}$$

□

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

Lemma 4.4.15 (e -copiables are orthogonal under multiplication)

$$\begin{array}{c} \text{Diagram A} \\ \text{Diagram B} \end{array} = \left\{ \begin{array}{ll} \text{Diagram C} & \text{if } \text{Diagram A} \neq \text{Diagram B} \\ \text{Diagram D} & \text{if } \text{Diagram A} = \text{Diagram B} \end{array} \right.$$

Proof.

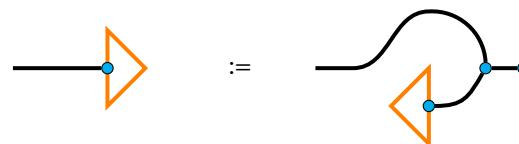
$$\begin{array}{ccccccc}
 \text{Diagram 1} & = & \text{Diagram 2} & = & \text{Diagram 3} & = & \text{Diagram 4} \\
 \text{Diagram 5} & = & \text{Diagram 6} & = & \text{Diagram 7} & = & \text{Diagram 8}
 \end{array}$$

So:

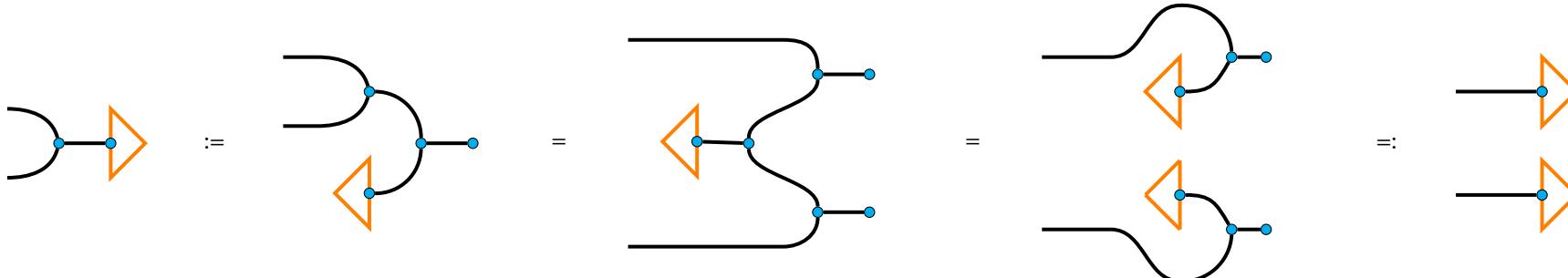
$$\begin{array}{c} \text{Diagram A} \\ \text{Diagram B} \end{array} \neq \begin{array}{c} \text{Diagram C} \end{array} \Rightarrow \begin{array}{c} \text{Diagram D} \end{array} = \begin{array}{c} \text{Diagram E} \\ \text{Diagram F} \end{array} = \begin{array}{c} \text{Diagram G} \end{array}$$

1

Convention 4.4.16 (Shorthand for the open set associated with an e -copyable). We introduce the following diagrammatic shorthand.



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



Lemma 4.4.17 (Co-match).

A string diagram showing a co-match operation. On the left, two parallel horizontal lines enter a red diamond (copy) and an orange diamond (delete). A black loop connects the output of the copy to the input of the delete. This is followed by an equals sign. To the right is a brace indicating two cases: if the inputs are unequal (red ≠ orange), the result is a black dot; if they are equal (red = orange), the result is a red diamond.

Proof.

A sequence of five string diagrams connected by equals signs, illustrating a proof. The first diagram shows a red diamond and an orange diamond with a black loop. Subsequent diagrams show the transformation of the loop and the diamonds through various stages until the final diagram, which is identical to the first.

The claim then follows by applying Lemma 4.4.15 to the final diagram. □

Lemma 4.4.18 (e-copyables are e-fixpoints).

Three string diagrams labeled (e-count), (coun/del), and (e-copy). The first diagram shows a red diamond with two outputs. The second diagram shows a red diamond with one output and a black line entering its other input. The third diagram shows a red diamond with one output and a black line exiting its other input. An equals sign connects the first and second diagrams, and another equals sign connects the second and third diagrams.

Proof.

A sequence of five string diagrams connected by equals signs, illustrating a proof. The first diagram shows a red diamond with two outputs. Subsequent diagrams show the transformation of the diamond and the lines through various stages until the final diagram, which is identical to the first.

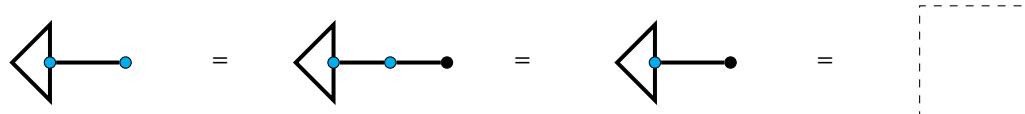
Observe that the final equation of the proof also holds when the initial e-copyable is the empty set. □

Lemma 4.4.19 (e -copiables are normal).



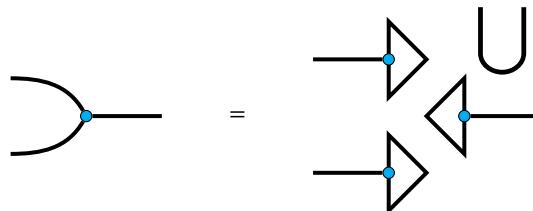
Proof.

(coun/del) (Lem. 4.4.18) (Prem.)

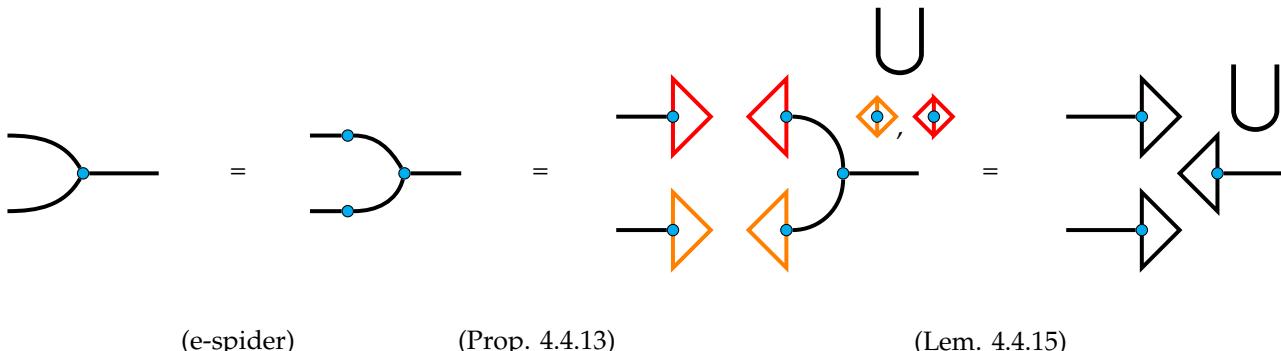


□

Proposition 4.4.20 (e -copyable decomposition of multiplication).

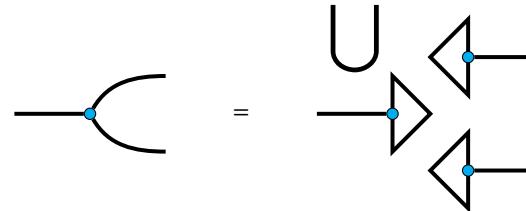


Proof.

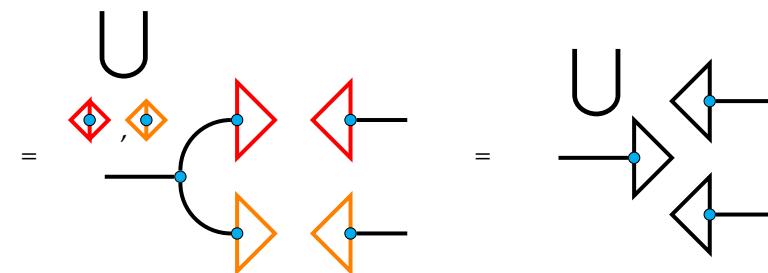
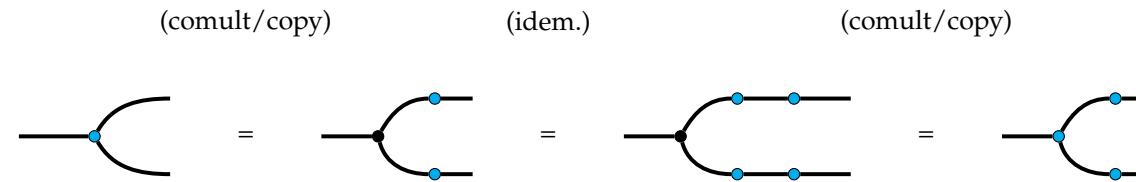


□

Proposition 4.4.21 (e -copiable decomposition of comultiplication).



Proof.



(Prop. 4.4.13)

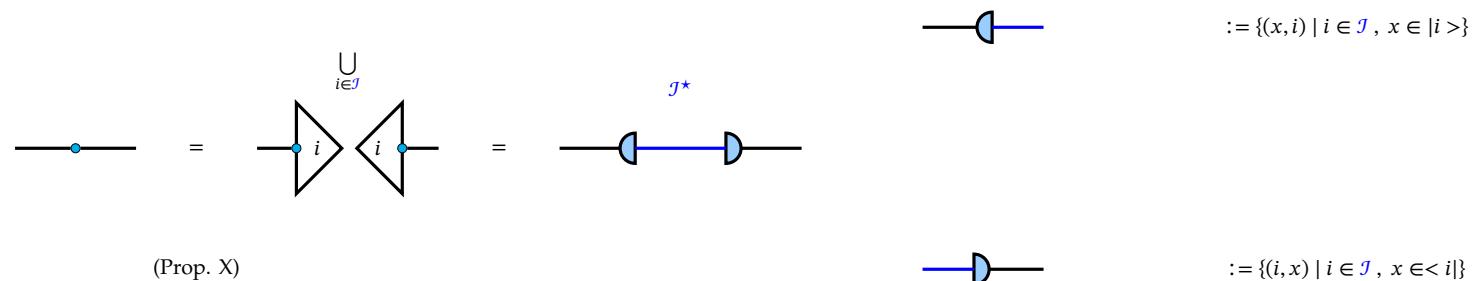
(Lem. 4.4.17)

A small, empty square box with a black border, likely a placeholder for a figure or diagram.

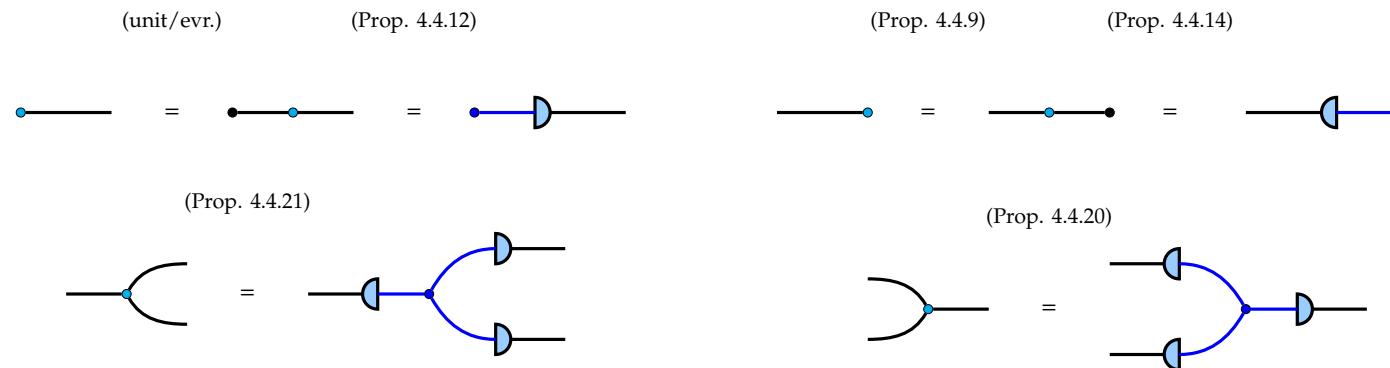
Now we can prove Theorem 4.4.8. First a reminder of the claim; we want to show that when given a sticky spider, the following relations hold if and only if the idempotent splits through a discrete topology.



The crucial observation is that the e -copyable decomposition of the idempotent given by Proposition 4.4.13 is equivalent to a split idempotent though the set of e -copyables equipped with discrete topology.



By copyable basis Proposition 4.4.12 and the decompositions Propositions 4.4.14, 4.4.20, 4.4.21, we obtain the only-if direction.



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

$$\begin{array}{ccccc}
 \text{(split)} & & \text{(Prop. 4.4.7)} & & \text{(split)} \\
 \bullet - \bullet \circlearrowleft & = & \bullet - \bullet \circlearrowleft \bullet \circlearrowright & = & \bullet \circlearrowleft \bullet \circlearrowright = \bullet
 \end{array}$$

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^* . 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^* , 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 .

$$\begin{array}{ccccc}
 \text{(split)} & & \text{(pfn.)} & & \text{(split)} \\
 \bullet - \bullet \circlearrowleft \bullet \circlearrowright & = & \bullet - \bullet \circlearrowleft \bullet \circlearrowright \bullet \circlearrowleft \bullet \circlearrowright & = & \bullet \circlearrowleft \bullet \circlearrowright \bullet \circlearrowleft \bullet \circlearrowright = \bullet - \bullet \circlearrowleft \bullet \circlearrowright
 \end{array}$$

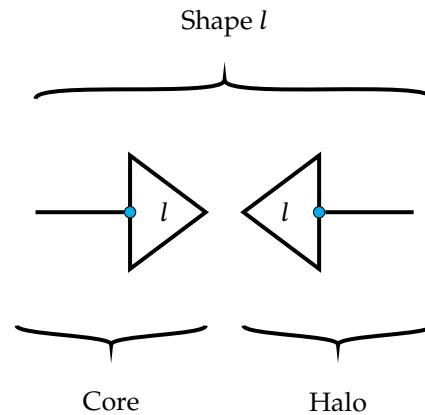
Definition 4.4.22 (Labels, shapes, cores, halos). Recall by Proposition 4.4.13 that we can express the idempotent as a union of continuous relations formed of a state and test, for some indexing set of *labels* \mathcal{L} .

$$\text{---} \bullet = \text{---} l \quad \text{---} l \quad \text{---}$$

$\bigcup_{l \in \mathcal{L}}$

The diagram shows a horizontal line with a blue dot representing a state. This is followed by an equals sign. To the right of the equals sign are two separate components. Each component consists of a horizontal line with a blue dot, followed by a diamond-shaped arrow pointing to the right, labeled 'l' at its center. These two components are separated by a small gap. Above the first component is a large union symbol (\bigcup) with the label ' $l \in \mathcal{L}$ ' underneath it.

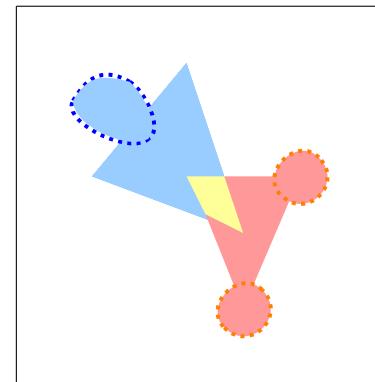
A *shape* is a component of this union corresponding to some arbitrary $l \in \mathcal{L}$. So we refer to a sticky spider as a labelled collection of shapes. The state of a shape is the *halo* of the shape. The halos are precisely the copiables of the sticky spider. The test of a shape is the *core*.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



Proposition 4.4.23 (Core exclusion: Distinct cores cannot overlap). *Proof.* A direct consequence of Lemma 4.4.17. □

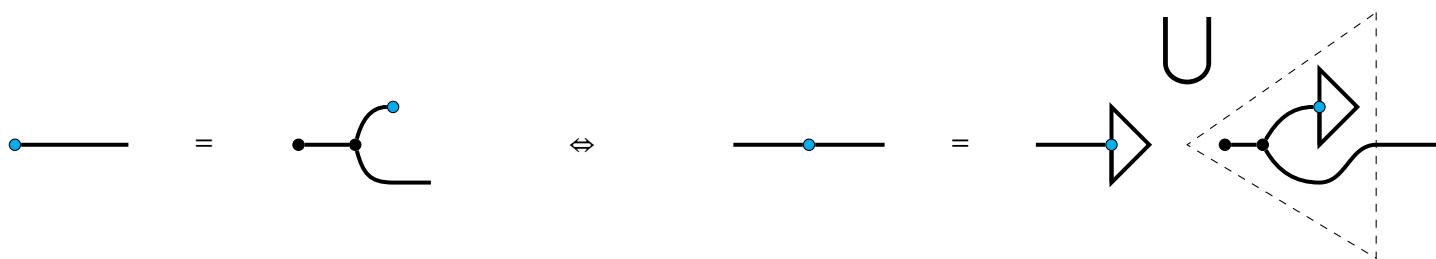
Proposition 4.4.24 (Core-halo exclusion: Each core only overlaps with its corresponding halo). *Proof.* Seeking contradiction, if a core overlapped with multiple halos, Lemma 4.4.18 would be violated. □

Proposition 4.4.25 (Halo non-exclusion: halos may overlap). *Proof.* By example:

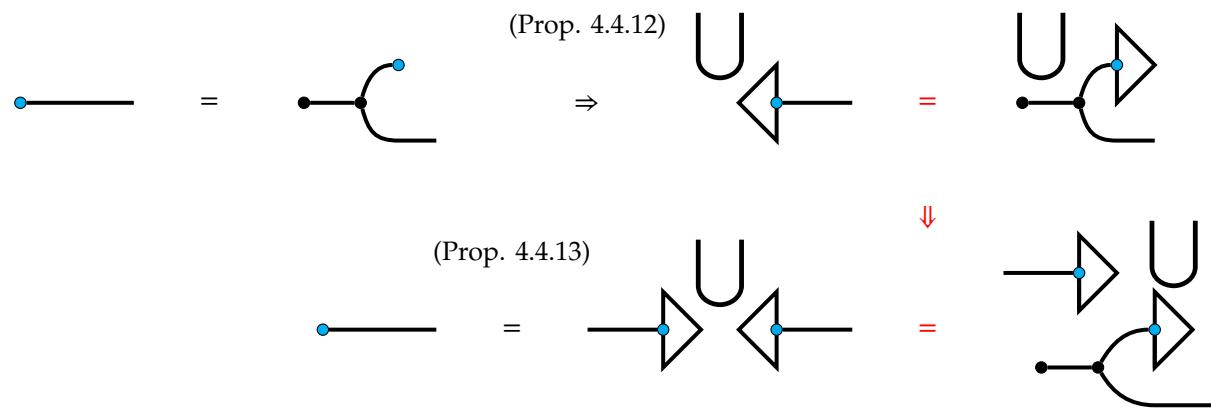


The two shapes are colour coded cyan and magenta. The halos are two triangles which overlap at a yellow region, and partially overlap with their blobby cores. The cores are outlined in dotted blue and orange respectively. Observe that cores and halos do not have to be simply connected; in this example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

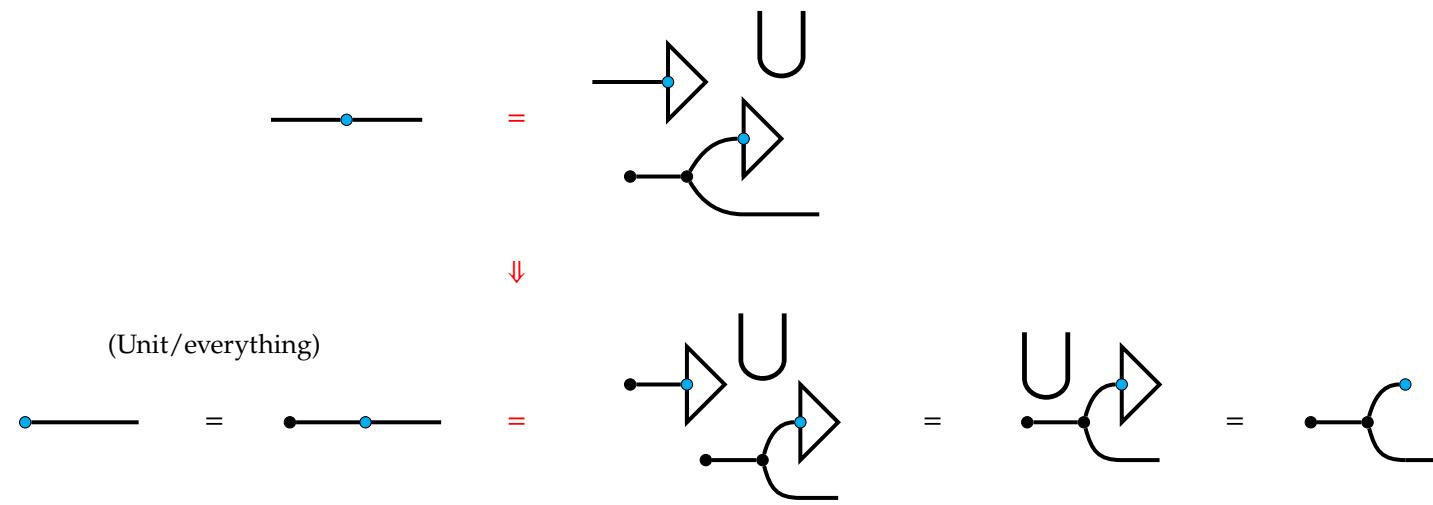
Corollary 4.4.26 (Only opens please). A sticky spider corresponds to a set-indexed disjoint collection of open sets when, in addition to the equations of Theorem 4.4.8, it satisfies one more, depicted below on the left.



Observe that the right hand equation above is precisely what we want expressed in diagrams: that every halo matches its core. For the forward direction, we have:



For the backward direction, we rely on the fact that cores are non-empty (or else we would fail to satisfy the identity equation of the split idempotent) to eliminate the floating scalars.



By Proposition 4.4.23, we have disjointness.

So, without loss of generality, we may treat any collection of disjoint open shapes on a page as a sticky spider.

5

Sketches in iconic semantics

How to reason formally with and about pictorial iconic representations as a semantics of natural language.

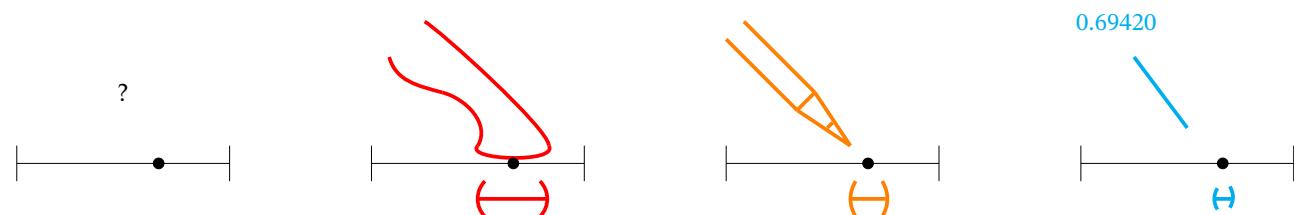
5.1 Preliminary concepts for the sketches

This section should be read as a smooth transition from the contents of the previous chapter towards sketches that gradually trade off rigour for expressivity, while ideally being descriptive enough that the reader trusts that the necessary details can be worked out.

5.1.1 Open sets: concepts

Apart from enabling us to paint pictures with words, **ContRel** is worth the trouble because the opens of topological spaces crudely model how we talk about concepts, and the points of a topological space crudely model instances of concepts. We consider these open-set tests to correspond to "concepts", such as redness or quickness of motion. Figure 5.1 generalises to a sketch argument that insofar as we conceive of concepts in (possibly abstractly) spatial terms, the meanings of words are modellable as shared strategies for spatial deixis; absolute precision is communicatively impossible, and the next best thing mathematically requires topology.

Figure 5.1: Points in space are a useful mathematical 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.

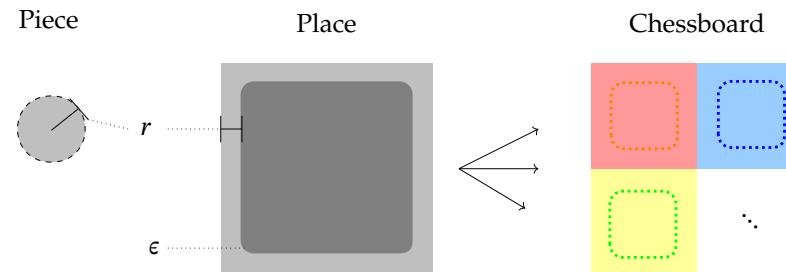


Maybe this explains the asymmetry of why tests are open sets, but why are states allowed to be arbitrary subsets? One could argue that states in this model represent what is conceived or perceived. Suppose we have an analog photograph whether in hand or in mind, and we want to remark on a particular shade of red in some uniform patch of the photograph. As in the case of pointing out a point on the real interval, we have successively finer approximations with a vocabulary of concepts: "red", "burgundy", "hex code #800021"... but never the point in colourspace itself. If someone takes our linguistic description of the colour and tries to reproduce it, they will be off in a manner that we can in principle detect, cognize, and correct: "make it a little darker" or "add a little blue to it". That is to say, there are in principle differences in mind that we cannot distinguish linguistically in a finite manner; we would have to continue the process of "even darker" and "add a bit less blue than last time" forever. All this is just the mathematical accommodation of a 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.

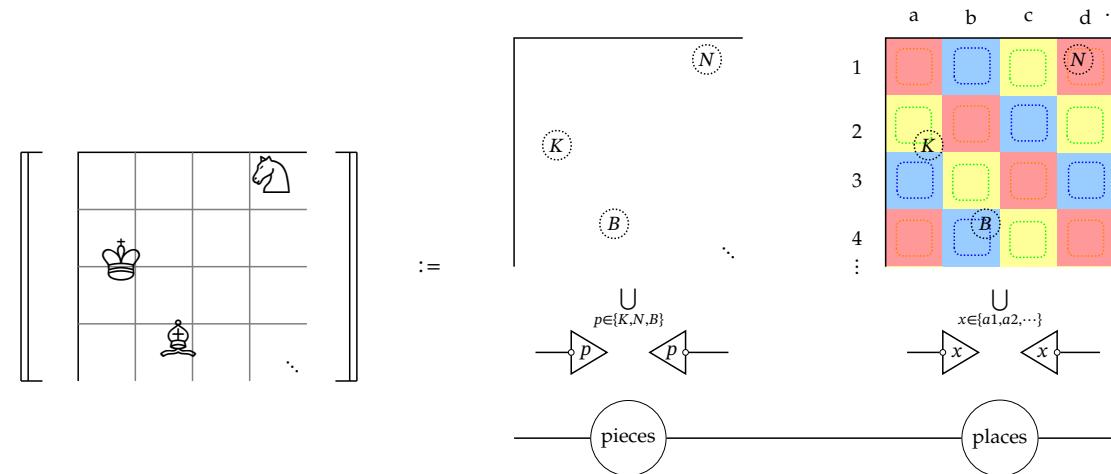
5.1.2 Using sticky spiders as location-tests

Example 5.1.1 (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.

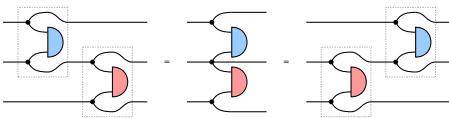


Now we observe that the calculation of positions corresponds to composing sticky spiders. We take the initial state to be the sticky spider that assigns a ball of radius r on the board for each piece. We can then obtain the set of positions of each piece by composing with the places spider. The composite (pieces;places) will send the king to a_2 , the bishop to b_4 , and the knight to d_1 , i.e. $\langle K \rangle \mapsto \langle a_2 \rangle$, $\langle B \rangle \mapsto \langle b_4 \rangle$ and $\langle N \rangle \mapsto \langle d_1 \rangle$. 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.



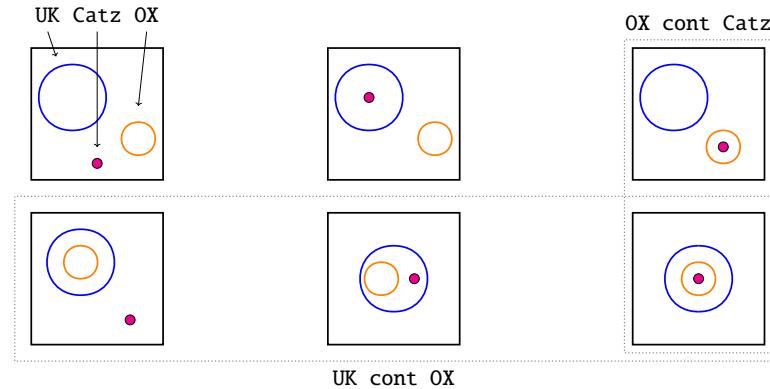
5.1.3 Copy: stative verbs and adjectives

Stative verbs are those that posit an unchanging state of affairs, such as Bob likes drinking. Insofar as stative verbs are restrictions of all possible configurations to a permissible subset, they are conceptually similar to adjectives, such as red car, which restricts permissible representations in colourspace. By interpreting conceptual spaces topologically, where concepts are particular open sets, we can test for whether states lie within concepts in the same way we can test whether a chesspiece is on a certain square. Moreover, **Con-tRel** conspires in our favour by giving us free copy maps on every wire, which allows us to define a family of processes that behave like stative restrictions of possibilities. These model stative verbs and adjectives. The desirable property we obtain is that in the absence of *dynamic* verbs that posit a change in the state of affairs, stative constructions commute in text.



If I'm just telling you static properties of the way things are, it doesn't matter in what order I tell you the facts because restrictions commute. Recall that gates of the following form are intersections with respect to open sets, and they commute. These intersections model conjunctive specifications of properties.

Example 5.1.2 (Containment and insideness).



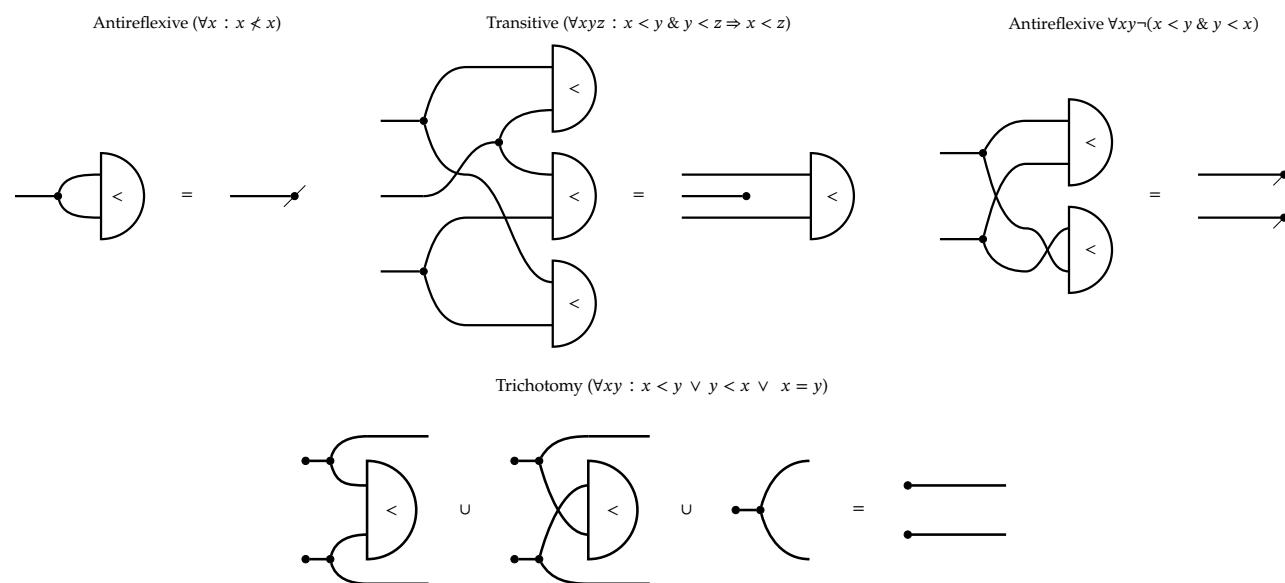
Consider the configuration space of a sticky spider on the unit square with three labelled shapes, which has 6 connected components, depicted. Oxford contains Catz. restricts away configurations where Catz is not enclosed in Oxford. Adding on England contains Oxford. further restricts away incongruent configurations, leaving us only with a single connected component, which contains all spatial configurations that satisfy the text. A similar story holds for abstract conceptual spaces, in which fast red car, fast car that is red, car is (red and fast) all mean the same thing.

5.1.4 The unit interval

To begin modelling more complex concepts, we first need to extend our topological tools. Throughout, we now consider string-diagrams to be expressions that may be quantified over, and we allow ourselves additional niceties like endocombinators. Ultimately we would like to get at the unit interval so we can do homotopies to move shapes around, which we plan to arrive at by first expressing the reals, and then adding in endpoints. However, 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 5.1.3 ([Fri05]). Let $((X, \tau), <)$ be a topological space with a total order. If there exists a continuous map $f : X \times X \rightarrow X$ such that $\forall a, b \in X : a < f(a, b) < b$, then X is homeomorphic to \mathbb{R} .

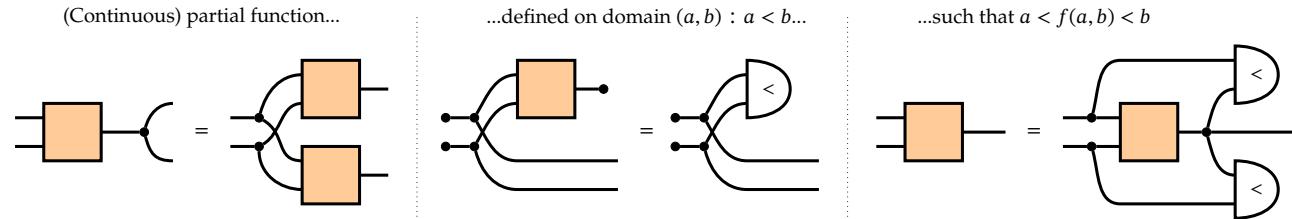
Definition 5.1.4 (Less than). We define a total ordering relation $<$ as an open set on $X \times X$ that obeys the usual axiomatic rules:



Definition 5.1.5 (Friedman's function). Just as a wire in **ContRel** has the discrete topology if it possesses spider structure (Proposition 4.4.3), a wire is homeomorphic to the real line by Theorem 5.1.3 if it possesses

Postscript: If you're already happy that in principle we may either start with nicer spaces or otherwise restrict ourselves to contractible opens, then you may skip the next two subsections and just glance at how relational homotopies differ from regular homotopies, and look briefly at the definition of a nice spider at the end. The relevant conceptual takeaway for the couple of sketches is that one may recover the usual topological notions such as simple connectivity, metrics and their open balls, and contractibility, from which one can in principle construct models of linguistic topological relations such as **touching**, **enclosure**, and so on. The submitted version of this thesis had detailed constructions of these linguistic topological relations "from scratch" but I've cut them, so only the next two sketches remain as artifacts to suggest that "low-level" hacking in **ContRel** is doable. I've opted to remove sketches of linguistic topological relations because (1) they took up too much space for too little gain (2) they still admitted counterexamples, and (3) it seems plausible that any analysis of linguistic topological primitives in mathematical terms will admit counterexamples, because I suspect they have the status of semantic primes [Wie96], which are characterised by their universality across languages and their unanalysability in simpler terms.

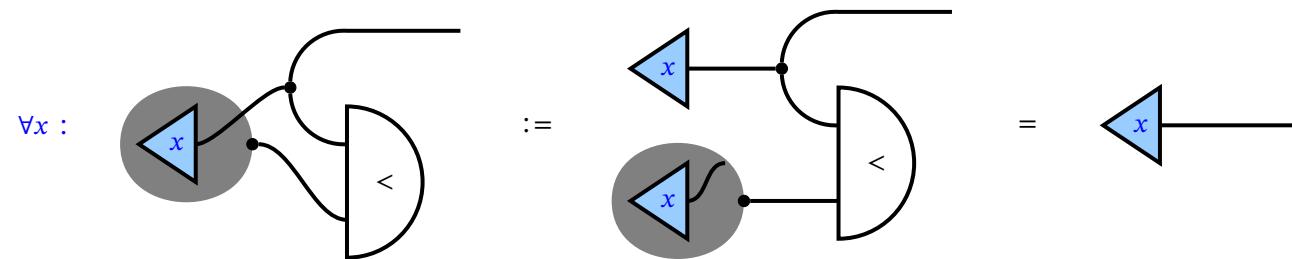
an open that behaves as Definition 5.1.4, and a map that satisfies:



Let's say that the unit interval is like the real line extended with endpoints. One way to define this that aligns with the usual presentation of the reals in analysis is to provide the ability to take suprema and infima of subsets, which are functions that map subsets to points. This kind of function is subsumed by a kind of structure on a category called an endocombinator.

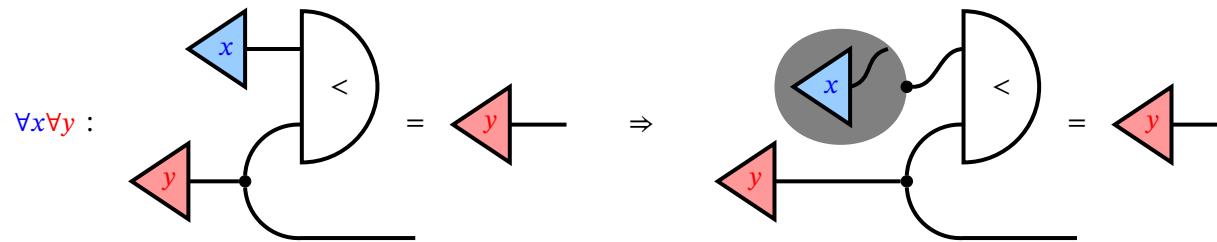
Definition 5.1.6 (Endocombinator). An *endocombinator* on a category \mathcal{C} is a family of functions on homsets typed $\mathcal{C}(X, Y) \rightarrow \mathcal{C}(X, Y)$, for all objects X, Y .

Definition 5.1.7 (Upper and lower bounds via endocombinators). Upper bounds are endocombinators that send states to points, which we depict as a little gray lassoed region around the state of interest. Recall that points are states with a little decorating copy-dot as they are copyable. The following equational condition quantified over all states characterises an "upper bound" endocombinator that returns an upper bound for any subset of a totally ordered space: in prose, such subsets are all less than their upper bound.



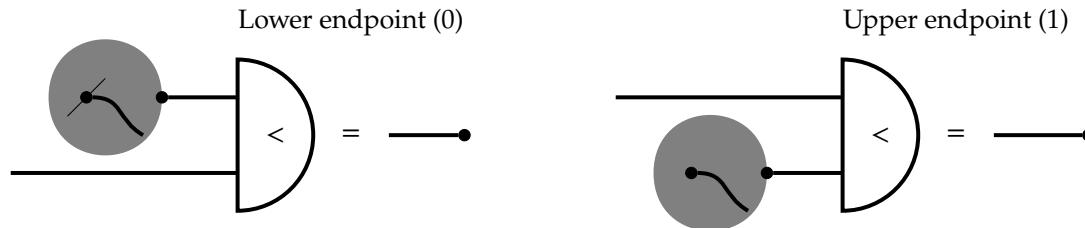
We can add in further equations governing the upper bound endocombinator to turn it into a supremum, or least-upper-bound.

Definition 5.1.8 (Suprema). An upper bound endocombinator is the supremum when the following additional condition (with caveats, see sidenote) holds: for all subsets y whose elements are all greater than those of a subset x , the supremum of x is less than all elements of y .



Now the lower endpoint is expressible as the supremum of the empty set, and the upper endpoint is the supremum of the whole set.

Definition 5.1.9 (Endpoints). The lower endpoint is the supremum of the empty state, and the upper the supremum of everything.



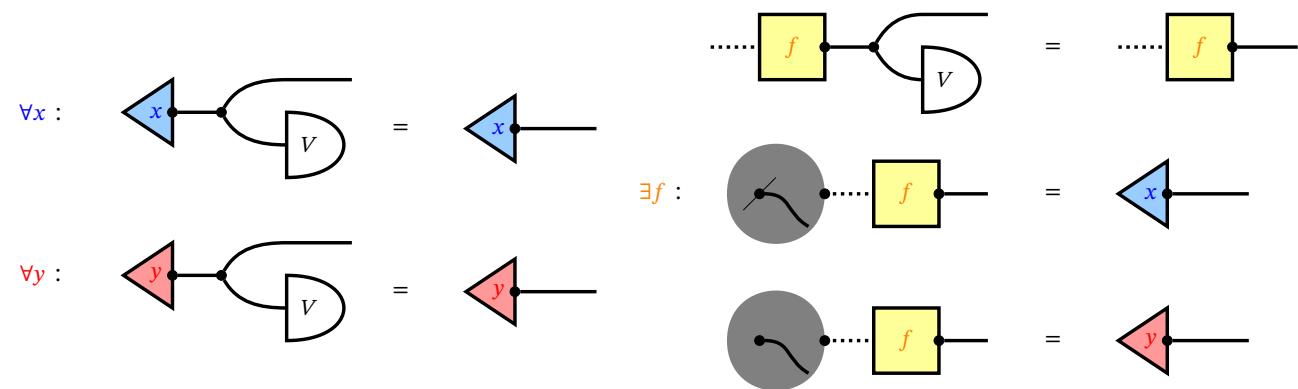
Definition 5.1.10 (The unit interval). In **ContRel**, an object equipped with a less-than relation (Definition 5.1.4), Friedman's function (Definition 5.1.5), and suprema (Definitions 5.1.7 and 5.1.8) is homeomorphic to the unit interval. Going forward, we will denote the unit interval using a thick dotted wire.

Unless y already contains $\sup(x)$, so the consequent of the implication needs a disjunctive case where $\sup(x) \cup y|_{\sup(x) <} = y$. The reason we cannot use \leq as an open (even though it would make this definition easier) is that it would imply the equality relation $=$ is an open, which would imply that the underlying space has the discrete topology, trivialising everything.

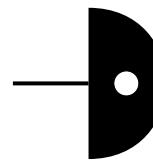
Conceptually, we are embedding the real line into a new space with two extra points, and then defining an extension of the less-than relation in terms of suprema to accommodate those points to characterise them as endpoints.

Example 5.1.11 (Simple connectivity). Recall that we notate points and functions with the same small black dot for copying and deleting, as points are precisely the states that are copy-delete cohomomorphisms. In prose, simple connectivity states that for any pair of points that are within the open V , there exists some continuous function from the unit interval into the space that starts at one of the points and ends at the other. The left pair of conditions state that the points x and y are within V . The right triple of conditions require the image of the homotopy f is contained in V , and that its endpoints are x and y .

V is simply connected when:

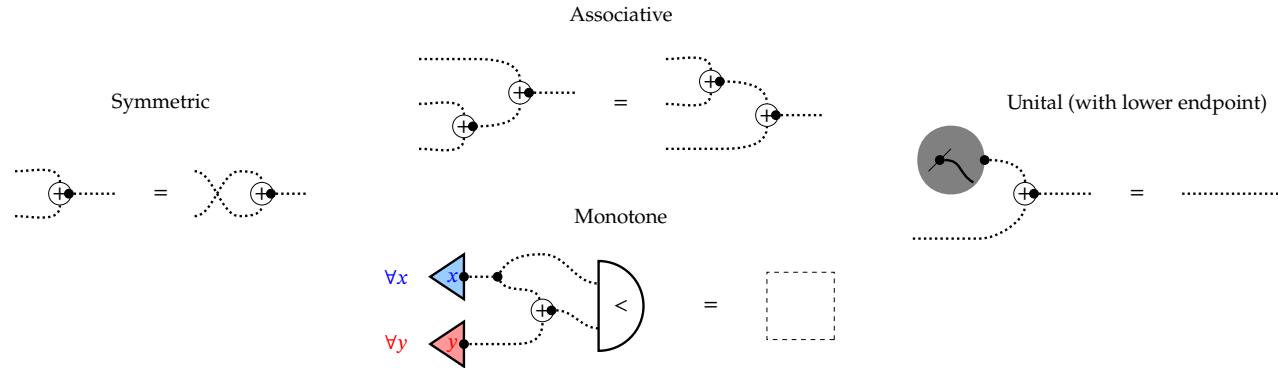


Simple connectivity is a useful enough concept that we will notate simply connected open sets as follows, where the hole is a reminder that simply connected spaces might still have holes in them.

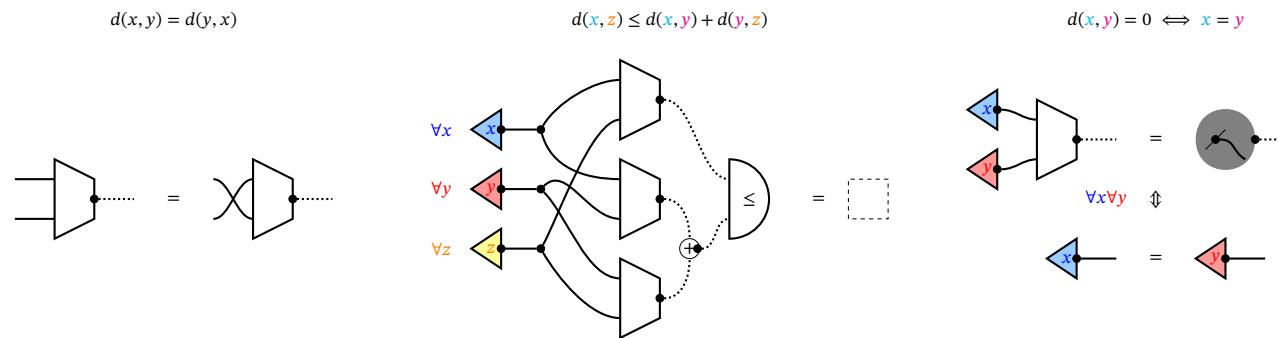


5.1.5 Metric structure

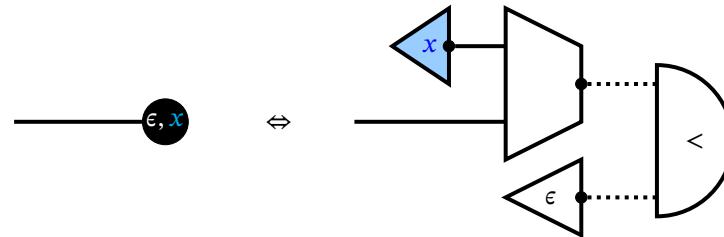
Definition 5.1.12 (Addition). In order to define metrics, we must have additive structure, which we encode as an additive monoid that is a function. All we need to know is that the lower endpoint of the unit interval stands in for "zero distance" – as the unit of the monoid – and that adding positive distances together will deterministically give you a larger positive distance.



Definition 5.1.13 (Metric). A metric on a space is a continuous map $X \rightarrow \mathbb{R}^+$ to the positive reals that satisfies the following axioms. We depict metrics as trapezoids because why not.



Example 5.1.14 (Open balls). Once we have metrics, we can define the usual topological notion of open balls. With respect to a metric, an ε -open ball at x is the open set (effect) of all points that are ε -close to x by the chosen metric.

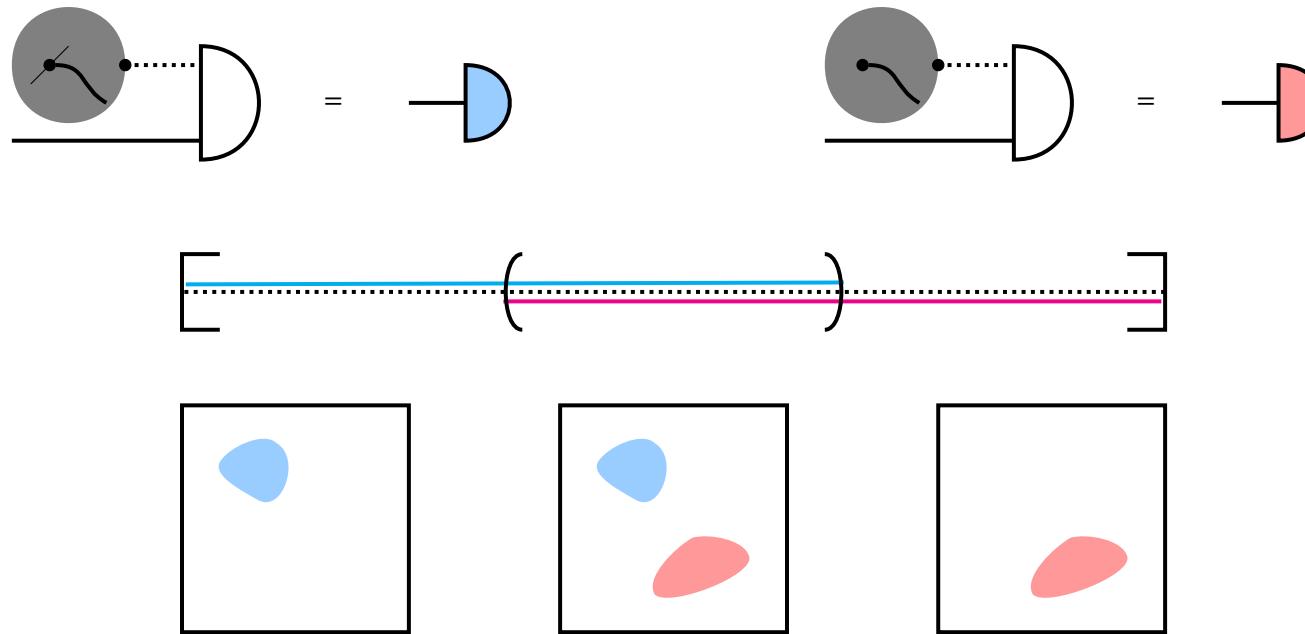


Open balls will come in handy later, and a side-effect which we note but do not explore is that open balls form a basis for any metric space, so in the future whenever we construct spaces that come with natural metrics, we can speak of their topology without any further work.

5.1.6 Relational homotopy

Definition 5.1.15 (Homotopy in **Top**). where f and g are continuous maps $A \rightarrow B$, a *homotopy* $\eta : f \Rightarrow g$ is a continuous function $\eta : [0, 1] \times A \rightarrow B$ such that $\eta(0, -) = f(-)$ and $\eta(1, -) = g(-)$.

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 a sticky spider expressing an open set in blue, which stays constant for a while. Then suddenly the ending open set in red appears (expressed by another sticky spider), and then the blue open disappears, and we are left with our ending; *technically* there was no discontinuity relative to the $[0, 1]$ -parameter in this relational homotopy between the two sticky spiders as endpoints, but there is something evidently discontinuous happening here that we would like to define away. 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 issue by asking for relational homotopies in **Con-tRel** to satisfy the additional condition that they are expressible as a union of "partial homotopies" that are individually total on $[0, 1]$.

Observe that the second condition asking for decomposition in terms of partial functions (of which total functions are a special case) comes for free by Proposition 4.2.19, as the partial functions form a topological basis. 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 . Definition 5.1.16 is "natural" in light of Proposition 4.2.19, that the partial continuous functions $A \rightarrow B$ form a basis for $\mathbf{ContRel}(A, B)$: we are just asking that homotopies between partial continuous functions – which can be viewed as regular homotopies with domain restricted to the subspace topology induced by an open set – form a basis for homotopies between continuous relations.

Definition 5.1.16 (Relational Homotopy).

$$\eta(0, -) = \textcolor{teal}{f}(-)$$

$$\eta(1, -) = \textcolor{magenta}{g}(-)$$

η is the union of homotopies of partial cts. maps

$\forall p$

$\dots\dots$
=

$$\bigcup_{p \in \text{Pfn}} \quad \dots\dots$$
 $\dots\dots$

$\dots\dots$
=

$\dots\dots$
 $\dots\dots$

$\dots\dots$
=

$\dots\dots$
 $\dots\dots$

$\dots\dots$
=

$\dots\dots$
 $\dots\dots$

p is a partial map

p is total on $[0, 1]$

5.1.7 Coclosure: adverbs and adpositions

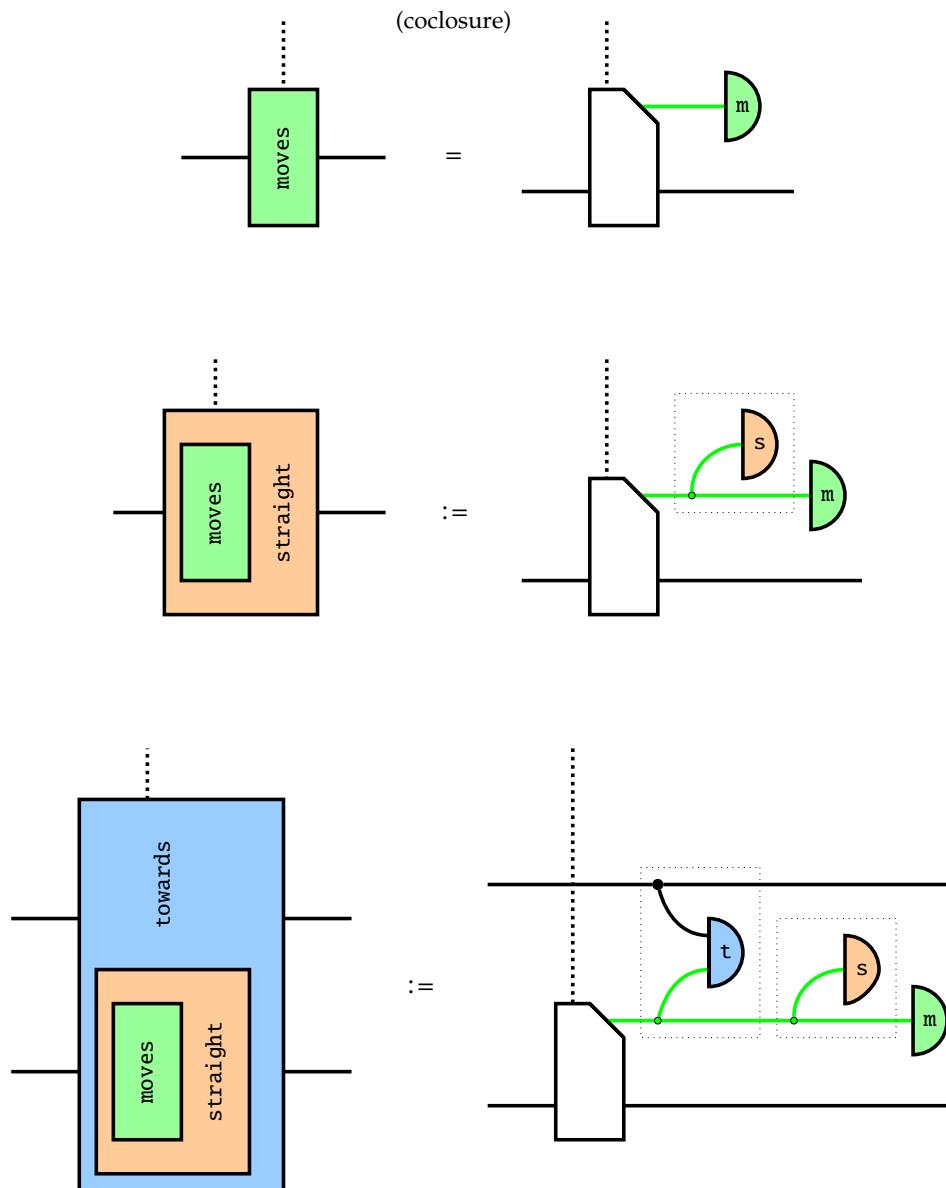


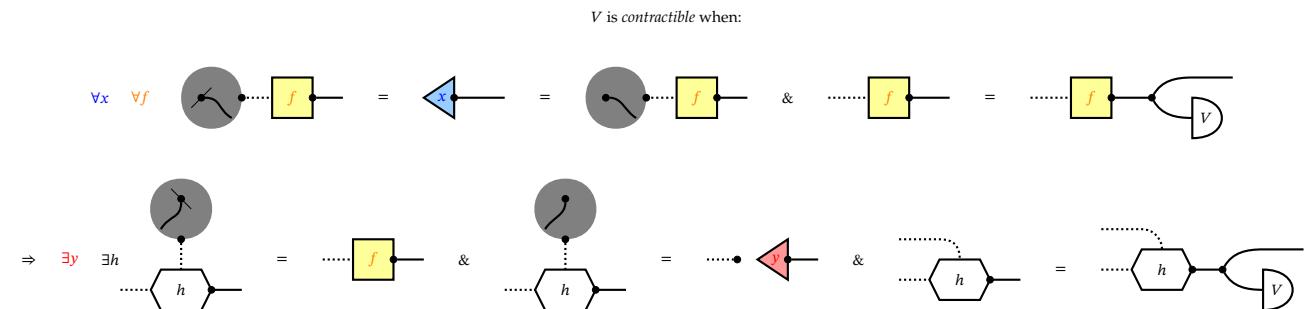
Figure 5.2: Recall that **ContRel** is coclosed (Proposition 4.3.22), which means that every dynamic verb may be expressed as the composite of a coevaluator and an open set on the space of homotopies. For instance, *move* is an intransitive dynamic verb, which corresponds to a concept in the space of all movements.

Figure 5.3: Adverb-boxes may be modelled as static restrictions in movement-space. For instance, *straight* may restrict movements to just those that satisfy some notion of path-length minimality: e.g., given a metric in movement-space on path-lengths, we may construct an open ball (Definition 5.1.14) around the geodesic to model the adverb *straight*.

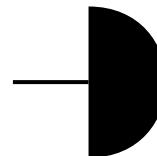
Figure 5.4: Similarly, adposition-boxes may be modelled as static restrictions on the product of the spaces of nouns and verbs. For instance, *towards* may be modelled as an open set that pairs potential positions of the thing-being-moved-towards with movements in movement-space that indeed move towards the target.

5.1.8 Nice spiders

Example 5.1.17 (Contractibility). 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. I've depicted homotopies here as hexagons for no particular reason, and they've got a dot to indicate that they're functions. In prose, for all points x and paths f such that f starts and ends at x and is contained within V , contractability implies that there exists a point y in h and a regular homotopy h that begins with f and finishes at the point y , and all of the images of the homotopy are contained within V .

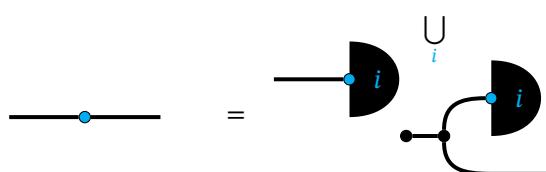


Contractible open sets are worth their own notation; a solid black effect, this time with no hole.



Let's assume for simplicity that henceforth, unless otherwise specified, we only deal with *nice* sticky-spiders where cores and halos agree and are both contractible opens; i.e. the spider can be expressed as a finite union of open solid blobs as effects followed by the same open solid blob as a state.

Definition 5.1.18 (Nice sticky-spiders). A sticky-spider is *nice* if it is equal to a union of contractible open effects followed by the same contractible open expressed as a state.



5.2 Composition of dynamic verbs via temporal anaphora

Dynamic verbs in iconic semantics may be modelled by homotopies, but non-parallel composition of homotopies is only defined up to parameters with indications of how the two separate homotopies begin and end relative to one another; i.e. temporal data.

Example 5.2.1 (Gluing homotopies sequentially at a time $\gamma \in (0, 1)$). Given two homotopies $f : X \times [0, 1] \rightarrow Y$ and $g : Y \times [0, 1] \rightarrow Z$, we may define their composite along Y with respect to $\gamma \in (0, 1)$ by compressing f to occur within $[0, \gamma]$ while holding g fixed at time 0, followed by compressing g to occur within time $[\gamma, 1]$ while holding f fixed at time 1.

$$f;_\gamma g(x, t) := \begin{cases} g(f(x, \frac{t}{\gamma}), 0) & \text{if } t < \gamma \\ g(f(x, 1), 0) & \text{if } t = \gamma \\ g(f(x, 1), \frac{t}{\gamma} - 1) & \text{if } t > \gamma \end{cases} : X \times [0, 1] \rightarrow Z$$

In this case, composition asks for one free parameter γ , but it is easy to see that we may ask for more, corresponding to the free parameters of gaps, overlaps, and so on.

The technical difficulty I'd like to sketch a solution for is that while these parameters must be given as real numbers in the interval $[0, 1]$, temporal natural language underspecifies: e.g. in the utterance Bob drank, and then he slept he could have drank in the morning and then slept in the afternoon, or both in the evening, and so on. The easy solution is to have absolute temporal anchors, but we seem to get by with less, which appears to necessitate a possible-worlds approach. Arguably the theoretical minimum we require is a kind of algebra for temporal aspects as in the Yucatec Maya language [Boh09], so here I sketch an algebra for temporal anaphora in **ContRel** that only requires copy-delete along with the standard topology on \mathbb{R} obtained by the encoding of intervals as the open set $< : [0, 1] \times [0, 1]$. Then I'll show how this temporal data can be used to supply the information required for homotopy composition, which should indicate that **ContRel** is in-principle sufficiently expressive for dynamic iconic semantics for natural language, i.e. the interpretation of text as little moving cartoons.

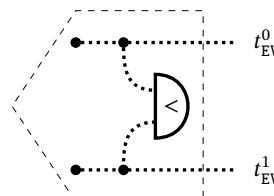
Definition 5.2.2 (A sketch text-circuit algebra for temporal anaphora). We consider three kinds of events.

The first is episodic, which corresponds to some interval on $[0, 1]$ with endpoints t_{EV}^0 and t_{EV}^1 . We model these as bipartite states with the initial constraint that $t_{EV}^0 < t_{EV}^1$. The second is habitual, which could in principle be an arbitrary subset of $[0, 1]$, but there are pathologies we would like to rule out as a matter of common sense (e.g. we don't really talk about events that occur in time according cantor set), so we treat habituals as open sets (unions of intervals) to be later constructed or supplied as constraints; when we are finished specifying the algebra, equipping it with unions as a kind of formal sum will approximate those open sets that are constructible by finite amounts of talking about times. The third is a hybrid of the first two, where

Postscript: These sketches are mostly a restructuring of content that otherwise dangled from the previous chapter. Dynamic verbs and modals are two new sketches I had in mind while initially writing the thesis but didn't make it to the submitted version. There will probably be technical errors, but the sketches are not intended to be rigorous. None of these sketches (and nothing else in this thesis for that matter) should be taken as canonical once-and-for-all solutions to the conceptual problems they are meant to tackle; they are more meant to provoke as first-pass attempts, and they are meant to demonstrate how to play around and have fun in **ContRel** with string diagrams. I'll also note here that everything in **ContRel** is a kind of truth-conditional possible worlds semantics (up to some arbitrary but fixed choice of what particular ensembles of shapes and movements the modeller supplies up front), so there are no guarantees about how any of this material would fare if one tried to take the diagrams and interpret them in terms of neural networks, and I make no claims about whether the mathematics reflects actual cognition. However, I will claim that these mathematical sketches reflect at least the phenomenology of how I think about language, which should come as no surprise because my methodology was armchair introspection.

we consider some open set with distinguished endpoints, modelled as a restriction/intersection of an interval with some other open set.

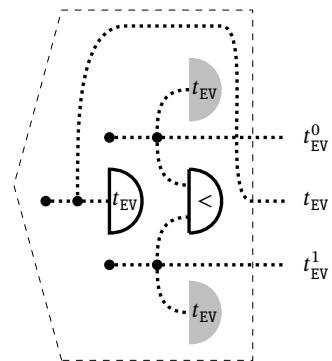
Episodic event
(Interval determined by ordered endpoints)



Habitual event (as constraint)
(An arbitrary open set on $[0, 1]$)

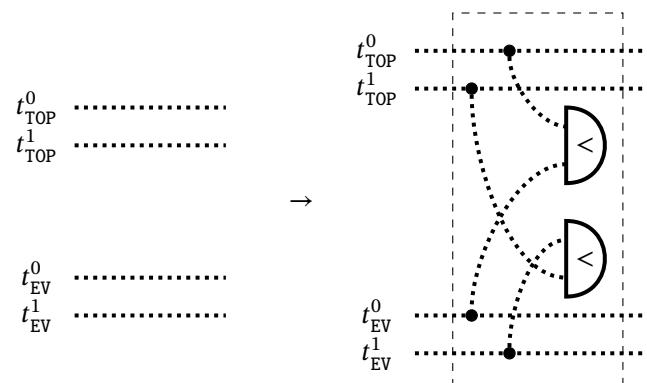


Hybrid event
(Open set with endpoints)



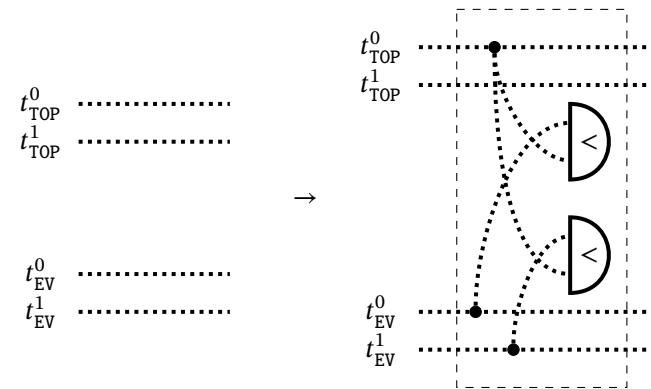
Now we model temporal aspects as circuit components — what appears to distinguish aspects from tenses is that aspects are always relative to the temporal data of two events, whereas tenses may be "intransitive" on events — so all of our aspectual data will involve constraining pairs of events (one of which is a TOPIC). The first kind of aspect we consider is *perfective*, which constrains an event time to be within topic time; we model this as imposing a constraint that the endpoints of the event must lie within the interval specified by the endpoints of the topic. In discourse, introducing a perfective constraint corresponds to adding a gate.

Perfective: $t_{EV} \subseteq t_{TOP}$
(Event time contained within topic time)



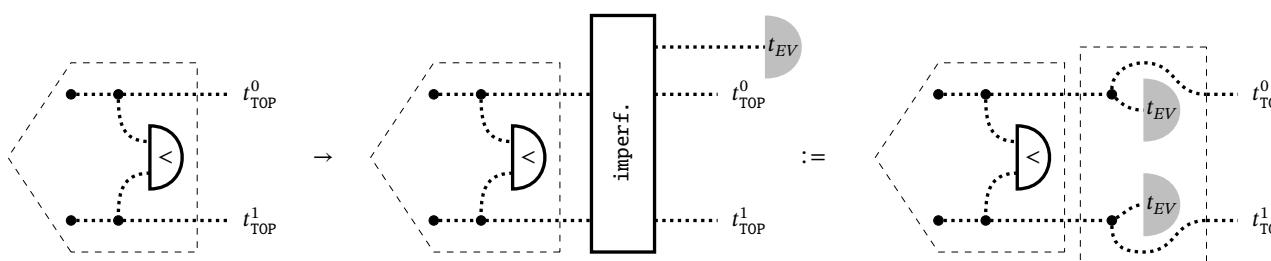
The *terminative* aspect constrains an event to occur entirely before the beginning of the topic time. Terminative composition of verbs may be glossed as (event) and-then (topic), and this kind of composition yields the view of text circuits as implicitly encoding the temporal order in which gate-as-events occur, where now the sequential ordering of gates matters. This failure of interchange interprets text circuits in something like a premonoidal setting [Jef98, RS24].

Terminative: $t_{EV} < t_{TOP}^0$
(Event will have been completed by the topic time)

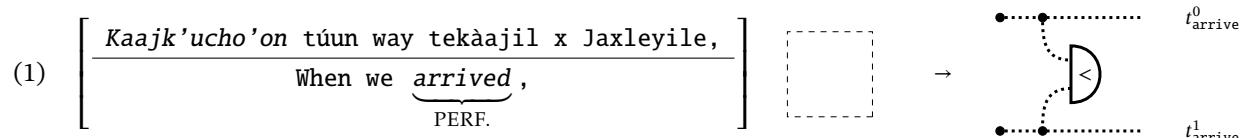


The *imperfective* aspect we consider as constraining an episodic topic time to lie within some ongoing habitual event, where the habitual event is represented as a free coparameter. In discourse, introducing an imperfective constraint corresponds to splicing in such a constraint, which we gloss as a gate that restricts the end-points of the topic interval to lie within the open set representing the habitual event time as a coparameter. We skip over the subtly distinct *progressive* aspect here as we won't need it for our later example, but it should be clear that an approach along these lines will also suffice.

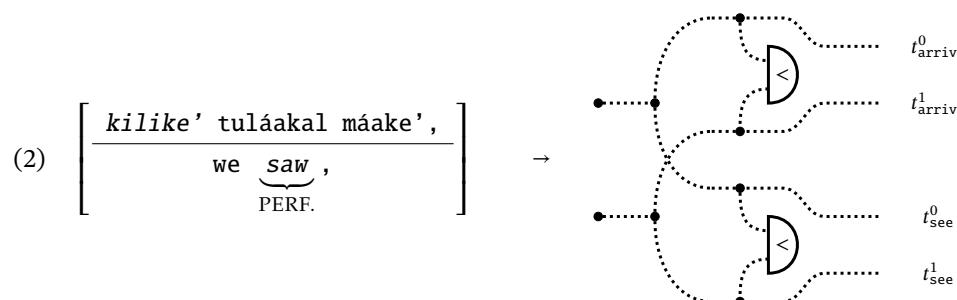
Imperfective: $t_{TOP} \subset t_{EV}$
(Episodic time contained within habitual event time)



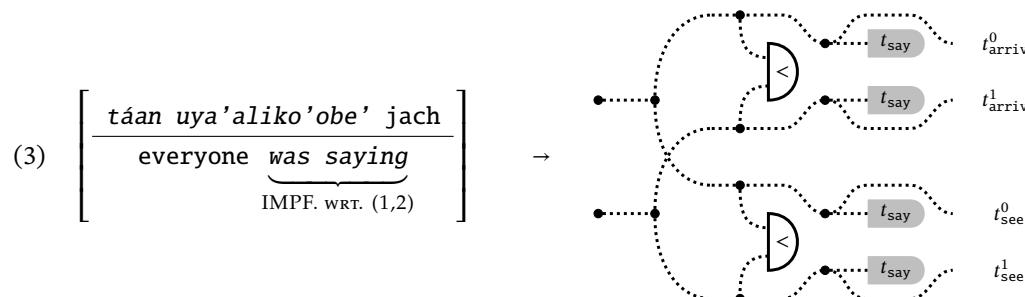
Example 5.2.3. So here is an example of Yucatan Maya taken from [Nat19], which is an excerpt of an interview with a speaker fleeing a cyclone. I have split the excerpt into numbered single-verb clauses, accompanied by glosses in English with aspect-markers and the corresponding evolution of a text-circuit by the discourse rewrites we have defined. The first event introduced into discourse is the arrival of the refugees in the village, which is marked as perfective.



The second event is what the refugees saw, implicitly concurrent with event (1), which we opt to treat with a prepended copy of endpoints. `arrive` & `see` then form an atomic topic for events (3) and (4), which we deal with by constraining both (1) and (2) in the same way. Note that there is a single variable open set t_{say} that is repeated 4 times in the diagram.



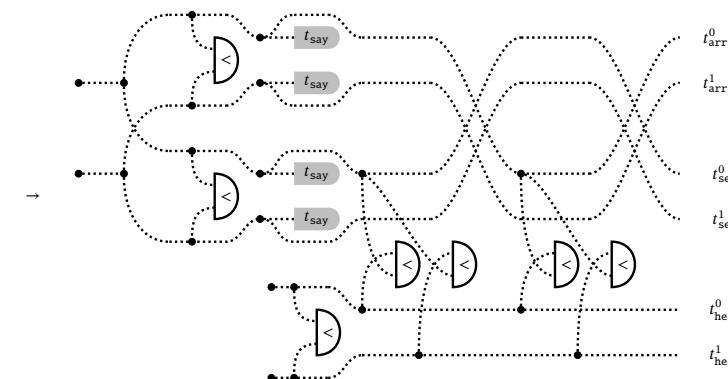
The third event refers to the villagers saying something, in the imperfective aspect with respect to events (1) and (2), so we constrain those topics accordingly. In gloss, it was an ongoing event that the villagers were saying something when the refugees arrived.



The fourth event refers to what the villagers had heard, in the terminative aspect with respect to (1) and (2).

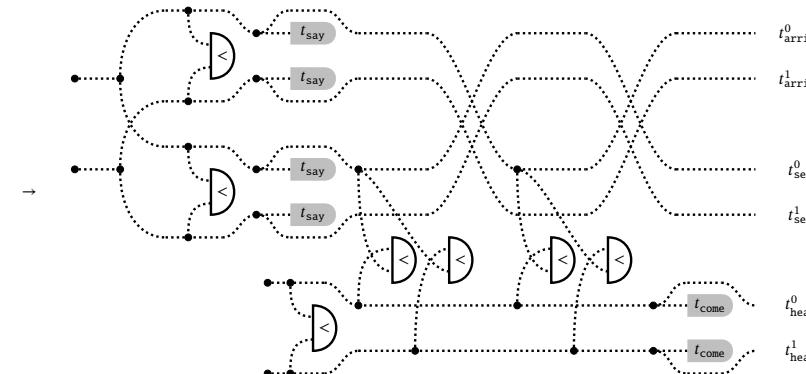
In gloss, the villagers were saying (reporting) the episodic event of them hearing something on the radio, and this hearing-event had completed before the refugees' arrival.

$$(4) \left[\frac{ts'uuyu'ubiko'ob ti' ràadyoe'}{(they) \underline{\text{had heard}} \text{ on the radio}} \right] \text{ TERM. WRT. (1,2)}$$



The fifth event refers to the coming of the cyclone, which was ongoing at the time of the villagers hearing the radio report. This introduces a new habitual event as the variable open set t_{come} , repeated twice in the diagram as constraints.

$$(5) \left[\frac{túun tàal le siklòono' .}{\text{the cyclone } \underline{\text{was coming.}}} \right] \text{ (4) IMPF. WRT. (5)}$$



Altogether, the final diagram represents a map from two open sets on $[0, 1]$ (representing the potentially habitual events say and come encoded as variable open sets t_{say} and t_{come}) to return a state in **ContRel** that encodes the set of possible endpoints for the episodic events arrive, see and hear: $\{(t_{arrive}^0, t_{arrive}^1, t_{see}^0, t_{see}^1, t_{hear}^0, t_{hear}^1)\}$. Moreover, we have set up the algebra to allow us to leverage compositional discourse structure in such a way that sampling any of the elements of the resultant set returns a choice of endpoints consistent with the temporal constraints of the excerpt.

5.3 Iconic semantics for modal verbs

In this sketch I want to deal with certain modal verbs: that means those of cognition and perception like to think and see, and the sketch will taper out towards some modal auxiliaries like wanting. These kinds of verbs are roughly characterised as requiring copies of entities to be instantiated in worlds similar to but not exactly that of whatever base narrative reality is referred to in the discourse. For example, in *Alice sees Bob drink a beer, Bob drinks another after Alice leaves.*, there are two Bobs, because the one in Alice's mental-theatre drinks a single beer, and the one in the base reality of the narration drinks two. So there are two worlds \mathfrak{W} here, one basic, and a \mathfrak{W}_A for the world in Alice's perception. Things get intractably tricky fairly quickly with these modals: to do epistemic logic means to have nested indices of what Alice thinks Bob thinks Alice thinks, to gossip is to reason about he-said-she-said, to understand complex narratives is to reason about stories-told-within-stories, and counterfactuals are a whole thing too. So that is a fundamental mystery: all this seems fairly complicated to encode and reason about symbolically, but it is phenomenologically fairly easy for adults to do, so what gives? What sort of mathematical presentation of these modals would at least reflect this lightness and ease?

I think thought-bubbles that show up in comic books are a pretty good start. Their cloudlike shape is a visual convention indicating a separate mental world, and they are typically used to represent want when the contents are also iconic representations.

Figure 5.5: Two examples by Mordillo, an artist I liked as a child: a thought bubble representing a woman, where the context of a stranded man implies a want for companionship, and a thought bubble representing a chair, where the context of a climber on a tall summit implies a want for rest.



The visual convention for cognitive and perceptive-alethic verbs is, as far as I can tell, a kind of x-ray effect into the contents of a head, which employs the familiar container metaphor: the head is a container for thoughts.

For alethic verbs in particular (those modals that are truth-preserving, in that they "do not forget" the truth), there's a need for the contents of the container to be synchronised with the contents of the outside world. Here are some observations that enable this in **Contrel**. The basic enabling insight is that, in Euclidean spaces, if we have a hollow container with a solid blob inside, there's an approximately continuous bijection between the (open set) insides of the container and the outside world.

Figure 5.6: On the left, a scene from the Simpsons showing the contents of Homer's mental-theatre. On the right, a depiction of two separate mental-theatres with a fisheye effect, taken from Steven Lahars "A Cartoon Epistemology" freely available online, which was also the initial inspiration for this sketch.



Figure 5.7: So the basic idea is to put representations of worlds inside bounded regions as containers, and in this way iconic semantics provides a univocal setting that displays all of the relevant worlds at once. We are free to pick visual conventions, as they are no more or less arbitrary than the assignment of indices and symbols such as \mathfrak{W}_A to the contents of possible worlds. Here is a sketch convention for containers on an iconic representation of a person for different modal verbs: seeing, thinking, feeling, owning, and wanting. I sent this excitedly with little supporting context to Bob while I was writing my thesis. He was concerned. Then I got concerned. Childlike became creepy, and neither are good looks. I think I have supplied enough context to make this sensible, but there's no way I'm going to beat the crazy allegations.

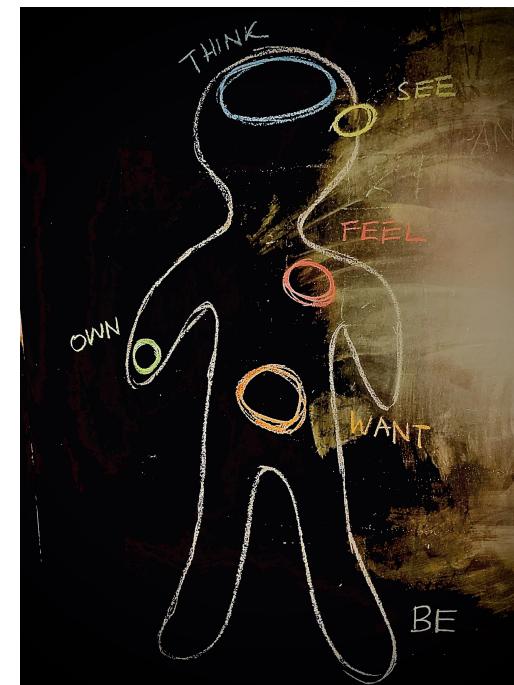
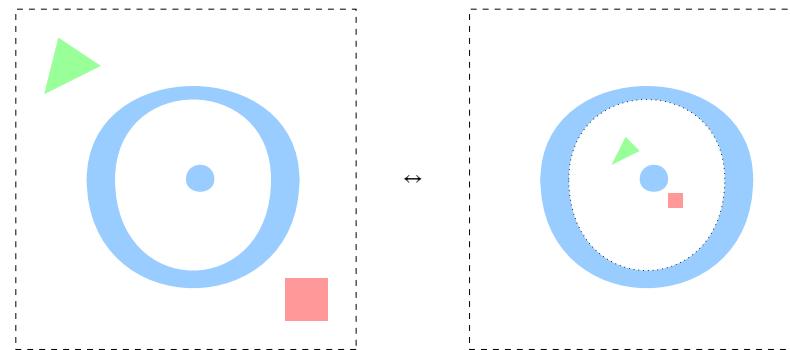
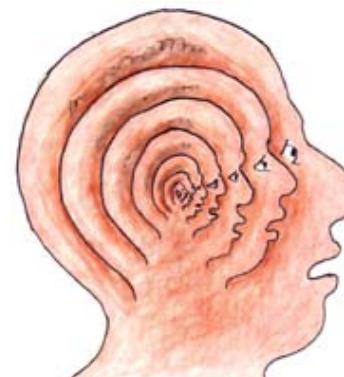


Figure 5.8: The inside and the outside of a container with a solid blob inside are both homotopic to the space with a puncture. This is only approximately a continuous bijection because the unbounded outside space can only map to the open interior of the container. We can use such bijections as a bridge to establish connections between elements of different possible worlds.



The second, and unfinished, idea is that if we have a handle on the individual components of sticky spiders, then we may use something like a very-well-behaved lens (hence its occurrence in the introduction) to ensure that the inside of the container is really behaving like a faithful storage medium for the goings-on outside. I think that's suggestive enough, and I'll deal with parthood in the next sketch. The last thing I want to deal with here is the problem of infinite regress for epistemic modals like knowing: if I know something, then I know I know it, and I know I know I know it, and so on. A naïve solution is to just use an infinitely-nested series of containers.

Figure 5.9: Again from Cartoon Epistemology, on the unsatisfactory nature of infinitely-nested containers: *But who is the viewer of this internal theatre of the mind? For whose benefit is this internal performance produced? Is it the little man at the center who sees this scene? But then how does HE see? Is there yet another smaller man inside that little man's head, and so on to an infinite regress of observers within observers?*



So the problem here is how to encode this infinite regress with finite means in an iconic model. The usual monadic approach still runs into the problem that you have to map a potential nested-infinity of possible worlds onto some finite model if one cares about cognitive realism. In iconic semantics, we can modify the space itself; here I think Escher was onto something.

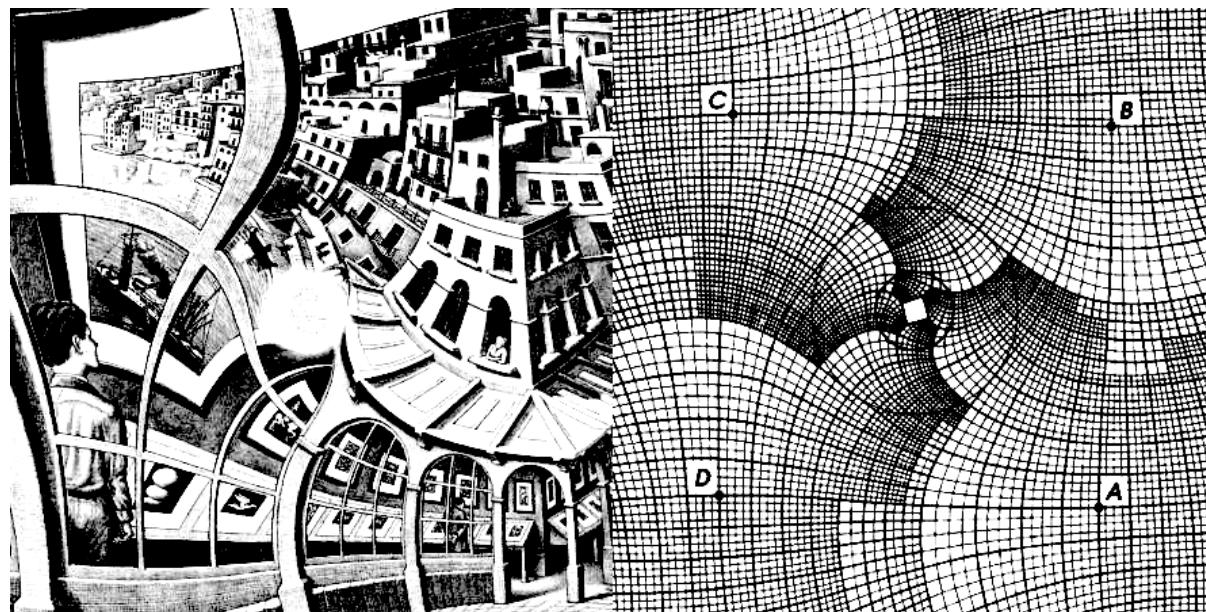


Figure 5.10: Escher's "Print Gallery" lithograph alongside his working sketch of the vortex-grid geometry the work was built on. On the left of the lithograph, an observer examines a framed painting of a town. Going clockwise, we see more details of the town, which has in it a print gallery, within which is the original observer. The missing centre of the piece where Escher signed the work obscures what would have been infinite nesting; the right-hand-side of the frame would have spiraled along the vortex infinitely. Treating the frame as a container, here we have an example of a container that contains itself, where movement clockwise indicates going down a level, clockwise going up, yet no explicit infinities anywhere.

The space in which such an arrangement can be realised is the same as that of the Penrose staircase: splitting the lithograph into four corners, each is a locally consistent snapshot, each gluing of quadrants is a consistent (as/de)scent, but the overall manifold obtained needs to be embedded in a higher dimension. While this in principle solves the problem of finitely representing infinite descent, these kinds of spaces are not grounded in physical, embodied intuitions. I think it is mathematically neat that there can exist topological models for such modal verbs, but whether such proposals are to be taken seriously as modelling cognition is a thorny matter I don't want to say more about.

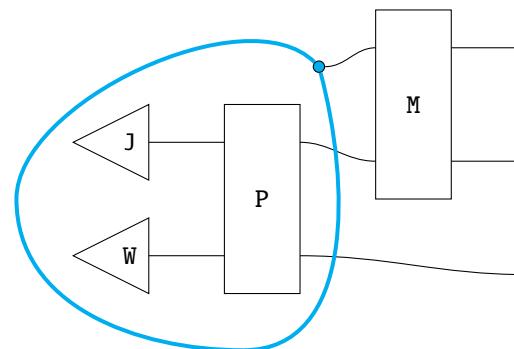
5.4 Iconic semantics for general anaphora via Turing objects

This sketch complements the sketch on modals, as it relies on the same container-trick. I would like to explain here how iconic semantics in **ContRel** might model untyped-boxes — these are conjunctions and verbs with sentential complements — as well as the more general linguistic phenomena of *entification* and *general anaphora* — where arbitrary discourse elements up to collections of sentences may be packaged up as if they were nouns and referred to. I suggest that the mathematical property of **ContRel** that enables this is that it contains **FinRel** equipped with a *Turing object*.

ENTIFICATION IS THE PROCESS OF TURNING WORDS AND PHRASES THAT AREN'T NOUNS INTO NOUNS. We are familiar with morphological operations in English, such as *inflections* that turn the singular *cat* into the plural *cats*, by adding a suffix *-s*. Another morphological operation generally called *derivation* changes the grammatical category of a word: for example, the adjective *happy* derives the noun *happiness*. With suffixes such as *-ness* and *-ing*, just about any lexical word in English can be turned into a noun, as if lexical words have some semantic content that is independent of the grammatical categories they might wear as a guise. With more complex discourse prefixes such as *the fact that*, we may also disguise sentences and text as nouns.

Example 5.4.1. Generalised anaphora as entification.

Jono is paid minimum wage. He didn't mind *it*.



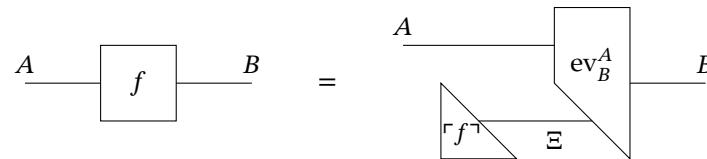
An example of entification. It may be argued that *it* refers to the *fact that Jono was paid minimum wage*. Graphically, we might want to depict the gloss as a circuit with a lasso that gives another noun-wire that encodes the information of the lassoed part of the circuit.

The problem at hand is finding an appropriate mathematical setting to interpret and calculate with such lassos. In principle, any meaningful (possibly composite) part of text can be referred to as if it were a noun. For syntax, this is a boon; having entification around means that there is no need to extend the system to

accommodate wires for anything apart from nouns, so long as there is a gadget that can turn anything into a noun and back. For semantics this is a challenge, since this requires noun-wires to "have enough space in them" to accommodate full circuits operating on other noun-wires, which suggests a very structured sort of infinity. Computer science has had a perfectly serviceable model of this kind of noun-wire for a long time. What separates a computer from other kinds of machine is that a computer can do whatever any other kind of machine could do — modulo church-turing on computability and the domain of data manipulation — so long as the computer is running the right *program*. Programs are (for our purposes) processes that manipulate variously formatted — or typed — data, such as integers, sounds, and images. They can operate in sequence and in parallel, and wires can be swapped over each other, so programs form a process theory, where we can reason about the extensional equivalence of different programs — whether two programs behave the same with respect to mapping inputs to outputs. What makes computer programs special is that on real computers, they are specified by *code*. Programs that are equivalent in their extensional behavior may have many different implementations in code: for example, there are many sorting algorithms, though all of them map the same inputs to the same outputs. Conversely, every possible program in a process theory of programs must have some implementation as code. Importantly, code is just another format of data. The process-theoretic characterisation of the code-wire in a process-theory of computation is this:

Definition 5.4.2 (Turing object). A *Turing object* Ξ in a process-theory is equipped with evaluation morphisms $\text{ev}_B^A : A \otimes \Xi \rightarrow B$ for all pairs of objects A, B such that for all morphisms $f : A \rightarrow B$, there exists a state $\lceil f \rceil_{I \rightarrow \Xi}$ of the Turing object such that partial evaluation with that state is equal to f . The diagrammatic convention and visual pun [Pav23] for such code-states and evaluators is to depict the state-triangle as if it is cut out from the rectangle of the evaluator.

$$\forall A, B \in \text{Ob}(\mathcal{C}) \exists \text{ev}_B^A \forall f \exists \lceil f \rceil$$



Any programming language is a model for text circuits, using the code-data format as the noun wire and Turing object. In **ContRel**, the unit square suffices as a Turing object for finite sets and relations, as we can use the container-trick of modals.

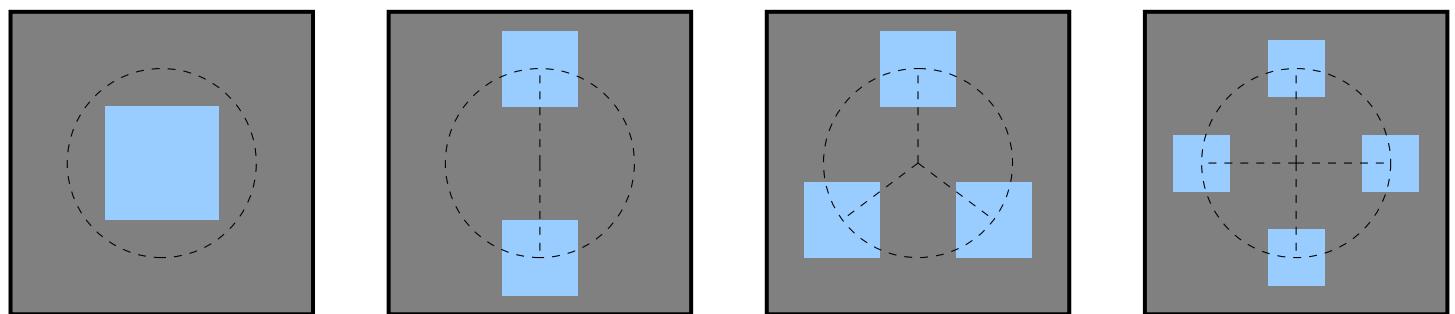
Proposition 5.4.3 (Sticky spiders on the open unit square model **FinRel** equipped with a Turing object). Using the open unit square with its usual topology as the Turing object, there is a subcategory of **ContRel** which behaves as the category of countable sets and relations equipped with a Turing object

Proof. By Construction 5.4.8, which we work towards. □

Another observation we could have made is that since computers really just manipulate code, every data format is a kind of restricted form of the same Turing object Ξ , but this turns out to be a mathematical consequence of the above equation (and the presence of a few other operations such as copy and compare that form a variant of frobenius algebra), demonstrated in Pavlovic's forthcoming monoidal computer book [Pav23], which is prefigured by a trilogy [Pav12, Pav14, PY18]. I would be remiss to leave out Cockett's work on Turing categories [CH08], from which I took the name Turing object. Both approaches to a categorical formulation of computability theory share the common starting ground of a special form of closure (monoidal closure in the case of monoidal computer and exponentiation in Turing categories) where rather than having dependent exponential types $\mathbb{A} \multimap \mathbb{B}$ or $\mathbb{B}^\mathbb{A}$, there is a single "code-object" Ξ . They differ in the ambient setting; Pavlovic works in the generic symmetric monoidal category, and Cockett with cartesian restriction categories, which generalise partial functions. I work with Pavlovic's formalism because I prefer string diagrams to commuting diagrams.

Lemma 5.4.4 $((0, 1) \times (0, 1))$ splits through any countable set X . For any countable set X , the open unit square \blacksquare has a sticky spider that splits through X^* — the discrete topology on X .

Proof. Proof by construction. Assume we work with nice spiders, so we only have to highlight the copyable 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 5.4.5 (Morphism of sticky spiders). A morphism between sticky spiders (here cyan and magenta) is any continuous relation that satisfies the following equation.

$$\text{Diagram showing a morphism between sticky spiders: } \text{cyan} = \text{magenta}$$

Lemma 5.4.6 (Morphisms of sticky spiders encode relations). For arbitrary split idempotents through A^* and B^* , the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \rightarrow B$.

$$\begin{array}{c} A^* \\ \text{---} \bullet \text{---} \square \text{---} \bullet \text{---} \text{---} \\ \forall \\ \text{---} \bullet \text{---} \square \text{---} \bullet \text{---} \text{---} \\ B^* \end{array} : \text{Rel}(A, B) \ni R \leftrightarrow \begin{array}{c} \text{---} \bullet \text{---} \square \text{---} R' \text{---} \square \text{---} \bullet \text{---} \text{---} \\ \simeq \quad \text{---} \bullet \text{---} R' \text{---} \bullet \text{---} \text{---} \\ = \quad \text{---} \square \text{---} R' \text{---} \text{---} \end{array}$$

Proof.

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

The diagram shows the equivalence of two string diagrams. On the left, a horizontal line with a blue dot enters a box labeled R' , which has a pink dot on its right side. This is followed by an equals sign. To the right of the equals sign is a sum symbol (\cup) above two morphisms. The first morphism consists of a blue diamond-shaped spider followed by an orange diamond-shaped spider. The second morphism consists of a blue square box followed by an orange square box. These two morphisms are connected by a horizontal line with a blue dot on the left and a pink dot on the right. Another equals sign follows, leading to a simplified form where the two morphisms are combined into a single blue diamond-shaped spider followed by an orange diamond-shaped spider, with a pink dot on the right side.

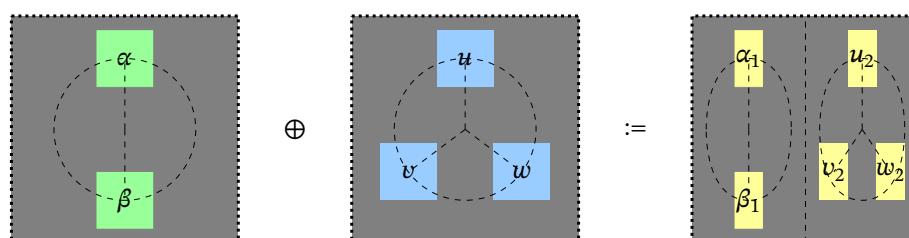
Since (co)copyables are distinct, we may uniquely reindex as:

This diagram shows the reindexing of the morphism from the previous step. The box R' is now labeled with a blue diamond-shaped spider and an orange diamond-shaped spider. An equals sign follows, leading to a sum symbol (\cup) above two morphisms. The first morphism is labeled $(a, b) \in R \subseteq A \times B$. The second morphism consists of a blue diamond-shaped spider labeled 'a' and an orange diamond-shaped spider labeled 'b'. The two morphisms are connected by a horizontal line with a blue dot on the left and a pink dot on the right.

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

This diagram shows the idempotence of relations. On the left, a sum symbol (\cup) above a morphism is followed by an equals sign. The morphism consists of a horizontal line with a blue dot entering a dashed box containing a blue diamond-shaped spider labeled 'a' and an orange diamond-shaped spider labeled 'b'. The dashed box is followed by another horizontal line with a pink dot. This is followed by another equals sign. To the right of the second equals sign is another sum symbol (\cup) above a morphism. The morphism consists of a horizontal line with a blue dot entering a dashed box containing a blue diamond-shaped spider labeled 'a' and an orange diamond-shaped spider labeled 'b'. The dashed box is followed by another horizontal line with a pink dot. A square symbol (\square) is located below the second dashed box.

Construction 5.4.7 (Representing sets in their various guises within \blacksquare). We can represent the direct sum of two \blacksquare -representations of sets as follows.



The important bit of technology is the homeomorphism that losslessly squishes the open 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.

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

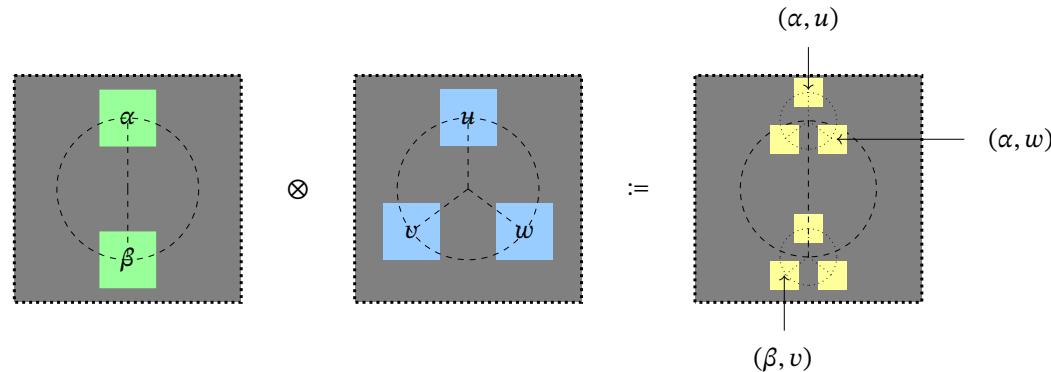
We express the ability of these relations to encode and decode the unit square in just either half by the following graphical equations.

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

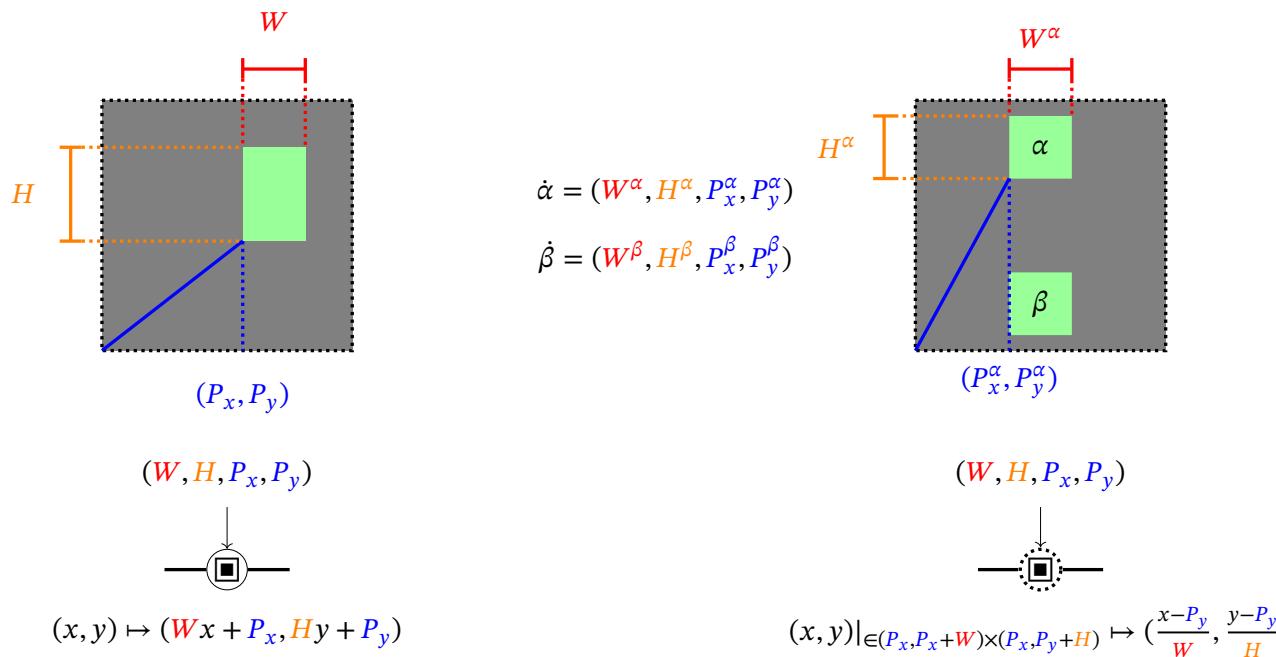
The following equation tells us that we can take any two representations in $\blacksquare\blacksquare$, put them into a single copy of $\blacksquare\blacksquare$, and take them out again.

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 \blacksquare parameterised by axis-aligned open boxes, that allow us to squish and stretch spaces. Thus for every representation of a set in \blacksquare by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a squish-stretch homeomorphism via the parameters of the open box, which we notate with a dot above the name of the element.

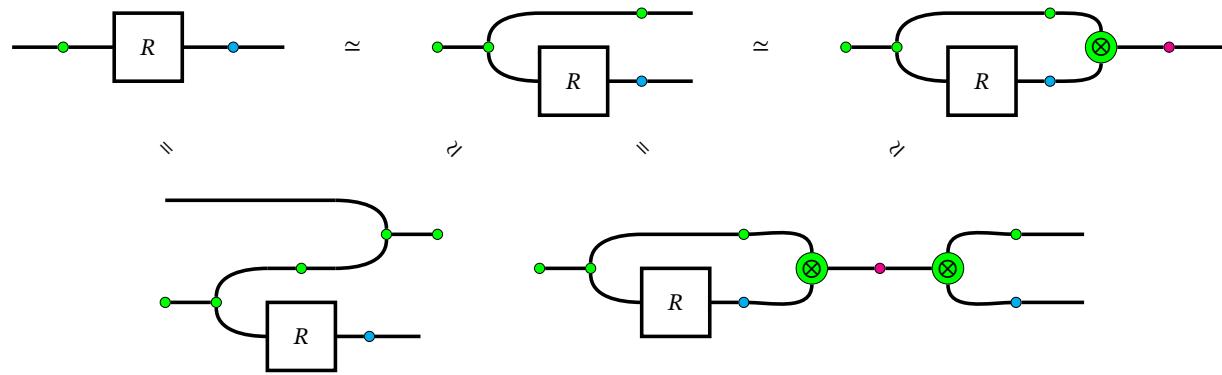


Now we can define the "tensor X on the left" relation $_ \rightarrow X \otimes _$ and its corresponding cotensor.

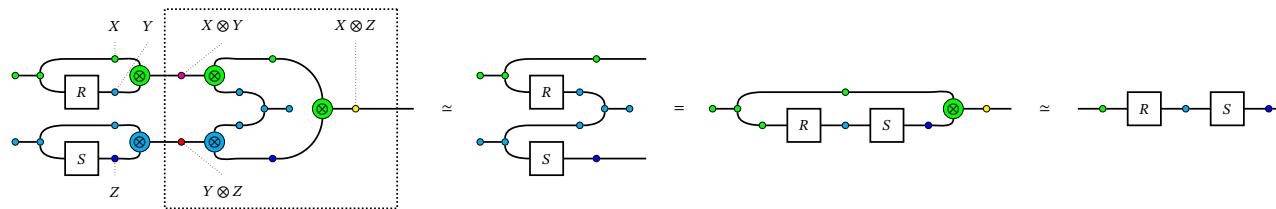
The tensor and cotensor behave as we expect from proof nets for monoidal categories.

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.

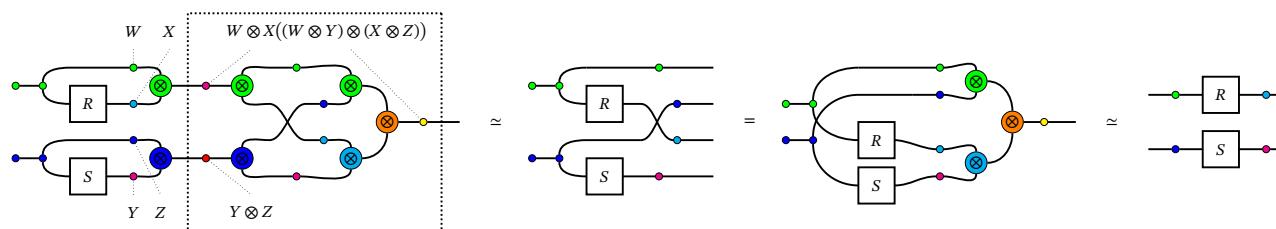
Construction 5.4.8 (Representing relations between sets and their composition within \blacksquare). With all the above, we can establish a special kind of process-state duality; relations as processes are isomorphic to states of \blacksquare , up to the representation scheme we have chosen. This is part of the condition for Turing objects. What remains to be demonstrated is that the duality coheres with sequential and parallel relational composition.



Under this duality, we have continuous relations that perform sequential composition of relations as follows.



And similarly, parallel composition. Therefore, we have demonstrated that the unit square behaves as a Turing object for the category of countable sets and relations.



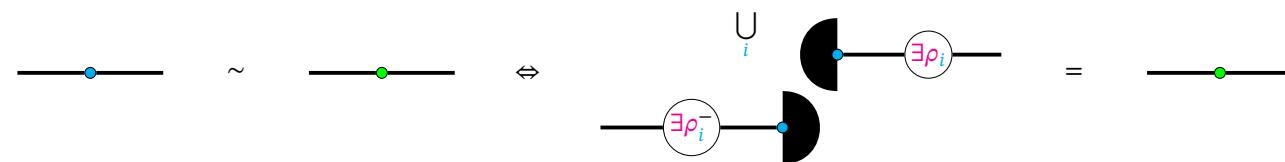
Configuration spaces and possible-worlds semantics favour working string-diagrammatically in **ContRel** over **Top**. The latter's cartesian monoidality limits it to one effect (delete) and only tensor-separable states, preventing native diagrammatic reasoning about correlated states analogous to entangled quantum states and spatial relations [CK17, WC21]. The Fregean notion of compositionality — knowing a composite system is equivalent to knowing all its parts — corresponds to tensor-separability in cartesian monoidal categories. Schrödinger's quantum mechanics insight offers an alternative: perfect knowledge of the whole doesn't necessitate perfect knowledge of the parts [Coe21]. Information about a composite system restricts possible outcomes a priori. The bell-state exemplifies this: we know both qubits measure identically, but discarding one qubit leaves maximal entropy for the remaining one. Similarly, imagining "a cup on a table in a room" entangles the objects' positions. Removing either object eliminates restrictions on the other's location, demonstrating that meaning resides in the entangled whole rather than individual parts.

5.5 Configuration spaces

Individual sticky-spiders correspond to static collections of set-labelled shapes in **ContRel**; in this sketch I want to talk about all the different ways the same collection of shapes can be arranged in space.

Let's also say we start with the ability to detect whether two sticky-spiders are related to one another by rigid displacements, expressed as a topological group with elements we denote ρ . Since sticky-spiders can be represented as unions of effects followed by states, we can define a binary relation on sticky-spiders that tells us whether they are the same up to rigidly displacing component shapes:

Definition 5.5.1 (Displacement relation). Two sticky-spiders (cyan and green, both assumed to be nice here), each with components indexed by I , are *equivalent up to displacement* when there exist ρ_i such that:



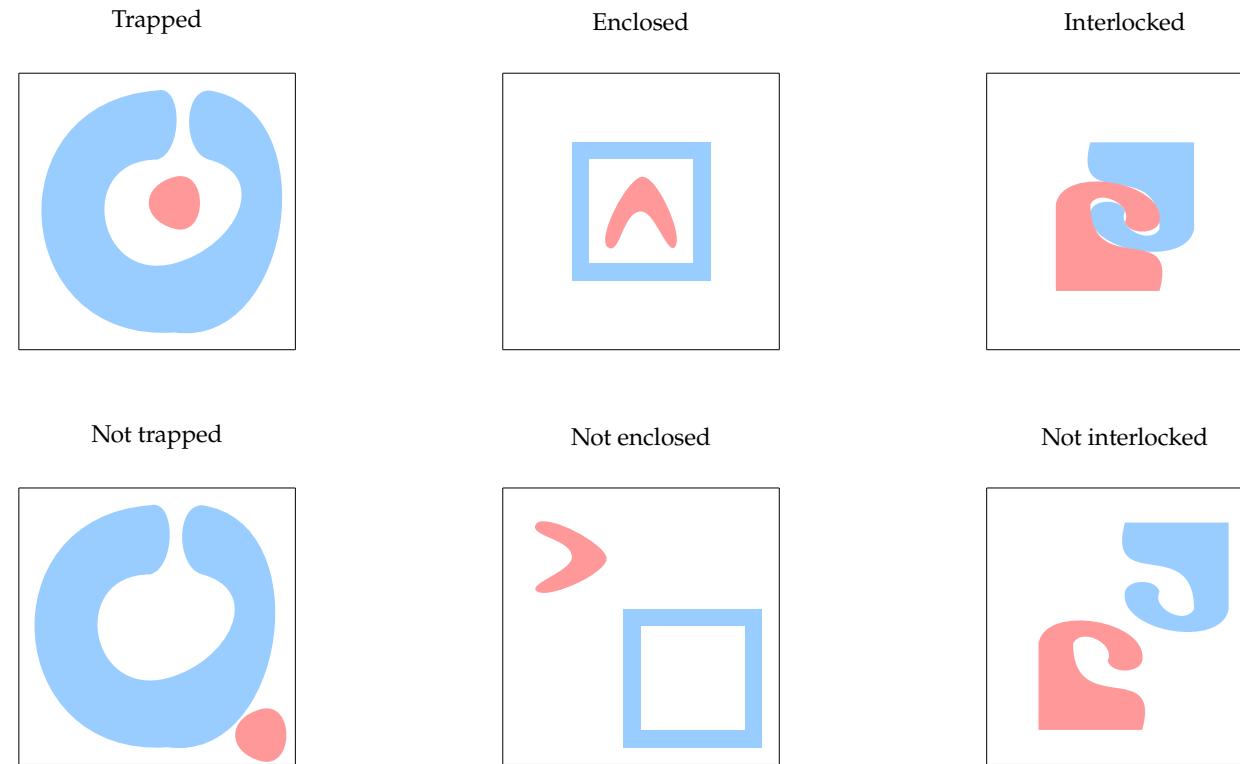
We've suppressed labelling of the states and we've contracted the cup to just depict the open state as a semi-circle.

Displacement is evidently an equivalence relation, and moreover requires that the two spiders related have the same number of components. Now given a particular nice spider, we treat its equivalence class of spiders as a configuration space in which we have access to all of its rigidly displaced variants at once.

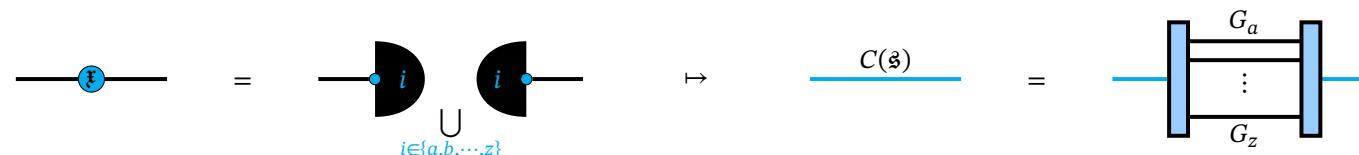
Definition 5.5.2. The *configuration space* $C(\mathfrak{s})$ of a nice spider \mathfrak{s} with indexing set I is the topological space with underlying set defined to be the equivalence class $[\mathfrak{s}]$ of \mathfrak{s} under displacement. Assuming the topological group of rigid displacements is itself a topological space G , the topology of $C(\mathfrak{s})$ is a restriction of $\bigtimes^{|I|} G$ to those $|I|$ -tuples of displacements witnessed by $[\mathfrak{s}]$.

Example 5.5.3 (The connected components of configuration space). Configuration space allows us to define a "slideability" relation between configurations of a spider \mathfrak{t} as the endpoints of continuous functions from the unit interval into $C(\mathfrak{s})$. This in turn allows us to consider what the connected components of configuration space are. Evidently, there are pairs of spiders that are both valid displacements, but not mutually reachable by sliding. For example, shapes might *enclose* or *trap* other shapes, or shapes might be *interlocked*. So at first blush, the connected components of configuration space tells us something about holes, or the cohomology of configurations. Depicted are some pairs of configurations corresponding to some linguistically topological terms that are mutually unreachable by rigid transformations, and so must live in disconnected components

of configuration space.

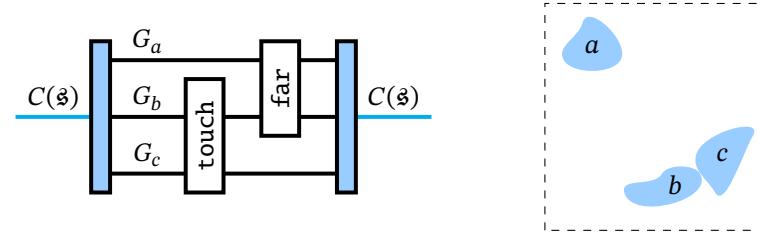


In configuration spaces we're making use of the fact that any displacement relationship comes with (up to a non-unique choice of basepoints for each component shape) a witnessing tuple of ρ_i s. As a consequence, the configuration space of a sticky-spider is a retract of the product space $\times^{|I|} G$ where G is the topological group of displacements, and we can use the identity relation between the section and retraction to strip the configuration space wire, revealing each of the $\times^{|I|} G$ like guitar strings: each element of the set that the initial nice spider \mathfrak{s} splits through gets its own string.

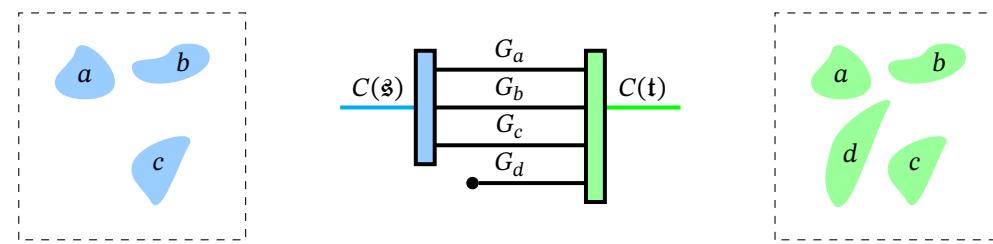


Note that although every guitar string is G , there is extra typing data indicating which element of the indexing set of the spider each G corresponds to. So here's a model in which the named wires of text circuits

make sense. We can put gates on the guitar strings, which may for example correspond to constraints on the relative positions of shapes in configuration space.



The next thing we can try is to add and subtract shapes from configuration spaces, and while there are technical details like matching choices of basepoints I'll gloss over, the gist is this: when the shapes in a nice spider \mathfrak{s} are a subset of the shapes in a nice spider \mathfrak{t} , we can add in states to the guitar-picture of \mathfrak{s} and wrap them up again using the idempotent of \mathfrak{t} , and we can delete wires in the guitar-picture of \mathfrak{t} and wrap that up using the idempotent of \mathfrak{s} .



The last stop in this sketch is disintegrating and integrating shapes; if we could freely break apart a shape, we know that in principle we get another configuration space where we can manipulate those parts, and if we can glue those pieces back together again, then we could do simple things like open and close containers. Let's first define the disintegration relation between spiders. Observe that the data of a nice spider is equivalently viewed as a function $f : I \rightarrow \mathfrak{O}$, where I is the indexing set, and \mathfrak{O} is some set of opens with whatever well-behaviour condition, along with the constraint that $f(x) \cap f(y) \neq \emptyset \Rightarrow x = y$ that enforces non-overlapping shapes. This perspective gives us a foothold to define a disintegration relation: a "more refined" spider is one that has a superset of I as domain, with a function that sends elements of the indexing set to either the same shape as f , or a subshape.

Definition 5.5.4 (Disintegration). Let \mathfrak{s} and \mathfrak{t} be nice spiders, described by functions $s : I \rightarrow \mathfrak{O}$ and $t : J \rightarrow \mathfrak{O}$ respectively. \mathfrak{t} disintegrates \mathfrak{s} ($\mathfrak{t} > \mathfrak{s}$) if there exists a surjective $d : J \twoheadrightarrow I$ such that $g = f \circ d$, and such that for all $i \in I$ and all $j \in d^{-1}(i)$, $g(j) \subseteq f(i)$.

Since the composition of surjectives is also surjective and the subsethood condition is transitive, disinte-

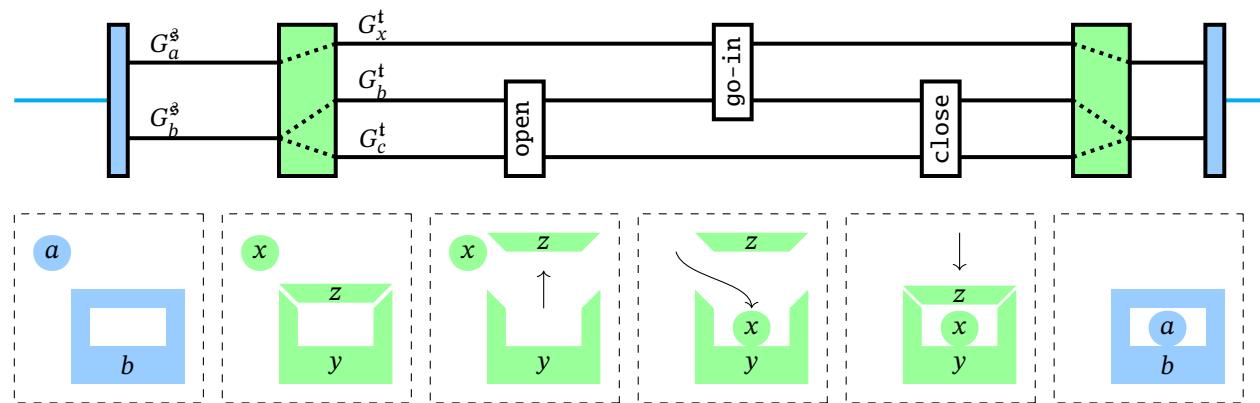
gration is a transitive relation. It's also reflexive, and since surjections $A \twoheadrightarrow B$ and $B \twoheadrightarrow A$ implies a bijection $A \simeq B$ and $X \subseteq Y$ with $Y \subseteq X$ implies $X = Y$, we also have antisymmetry, and hence a partial order. Treating the identity disintegration as globally minimal, we can define shatterings as locally minimal elements.

Definition 5.5.5 (Solve). t shatters s if $t > s$, and for all spiders q , $t > q > s \Rightarrow q = t$ or $q = s$, up to bijective relabellings of indexing sets.

The intuition behind shattering is that the \subseteq -condition in the disintegration relation lets the disintegrating spider "shave a little" off of the disintegrated spider, and locally minimal disintegrations "shave the least off", doing the best they can to partition shapes. So now we get gluing for free:

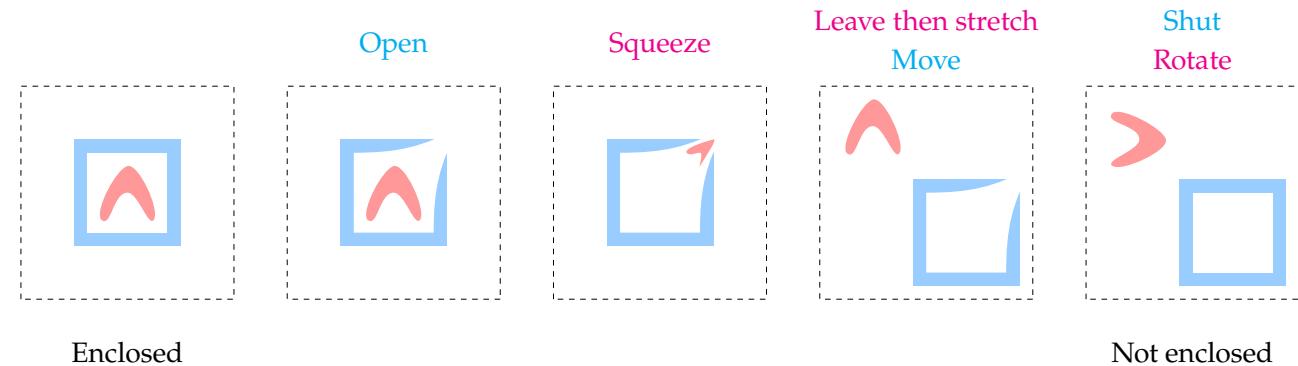
Definition 5.5.6 (et Coagula). t is a *gluing* of s if s shatters t .

Example 5.5.7 (Putting something in a container). To put a blob inside a container, we first shatter the container of the initial spider s to obtain a new spider t that expresses the container as a combination of a container and a lid, then (implicitly using dynamic verb composition of terminatives) we can move the lid, put the blob in, close the lid, and glue. Below the circuit we represent one possible series of consistent snapshots as a vignette, out of the many possible series of configurations that satisfy our linguistic description above.



In principle, shapes can be shattered arbitrarily finely, which permits us some degree of freedom in specifying how a container opens. In conjunction with a topological group of transformations that includes scaling, we may express different ways in which things get in and out of containers, or otherwise leave the original connected component of configuration space they start in. Here again I'm colour coding different shapes of

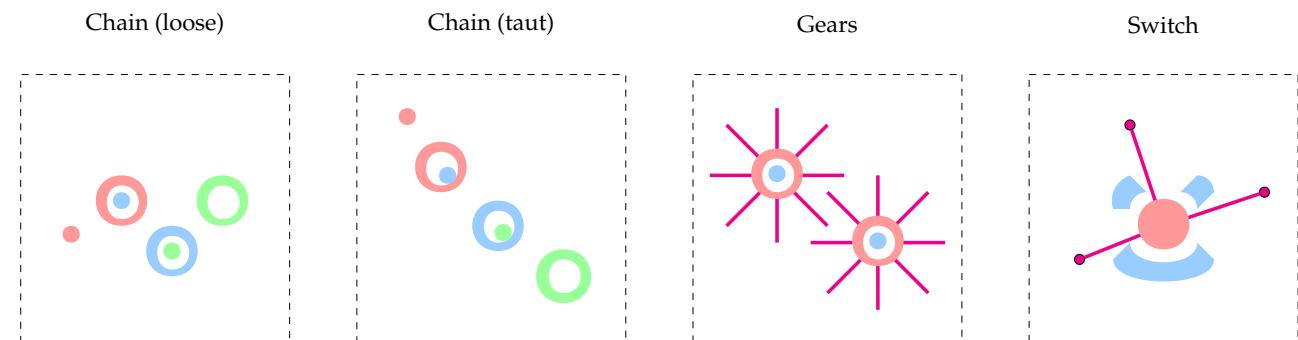
the same spider with different colours.



OBJECTION: ISN'T THIS WAY OUTSIDE THE SCOPE OF FORMAL SEMANTICS? Insofar as semantics is sensemaking, we certainly are capable of making sense of things in terms of mechanical models and games by means of metaphor, the mathematical treatment of which is concern of Section 5.6. It's probably the case that any definition that encompasses what's going on here as formal semantics would also have to consider the programming of a videogame to also be a form of formal semantics; personally I think that's ok, because I don't consider any particular form of mathematics-as-methodology to be privileged over others. Feel free to disagree.

I'll close this sketch with something cute: if manipulating shapes in configuration space is serious and sensible stuff, then just about anything is. We can (ab)use the fact that shapes of a nice sticky spider do not overlap to model mechanical components, where acceptable configurations of different shapes are mutually constrained in a productive way. In particular, this means we may consider any linguistic semantics grounded in mechanical or boardgame-tabletop models to be formal: in principle anything that can be represented by mechanisms and meeples is fair game. This gives us some cool possibilities for formal models of natural language, as there are a lot of mechanical models, including: clocks [duh.], analogues of electric circuits [Spi], computers [Ric15], and human-like automata [wik22].

Example 5.5.8 (Mechanical semantics). Here I'm going to allow shapes to be unions of disjoint contractibles, and I'll colour-code the different shapes in the spiders differently so the different components are clear:



5.6 Formal models of figurative language

Figurative language is when language is used non-literally, e.g. to bathe in another's affection. Figurative language subsumes analogy (built like a mountain), metaphor (she got a lot out of that lecture) and some idioms (raining cats and dogs). The issue with figurative language for formal semantics, insofar as formal semantics is concerned with truth-conditions, is that one requires an underlying model in order to begin truth-conditional analysis. The role of figurative language, especially that of metaphor, is in some sense to provide those models in the first place. The process by which language constructs the underlying model is essentially by structural correspondences, so the truth-theoretic (or inquisitive, or dynamic) approach to semantics operates at an inappropriate stage of abstraction. We might illustrate or depict schema to represent figurative language, but to the best of my knowledge, there is no formal account of how the systematicity of a chosen schematic corresponds to the organisation of a metaphor or concept. So what is required is a methodology to construct the underlying models from the figurative language in a more-or-less systematic way.

The whole point of mucking about with **ContRel** earlier is this: figurative language can be formally interpreted as vignettes involving topological figures. I will demonstrate here that cofunctors from **ContRel** into text-circuits representing utterances are promising candidates for the formalisation of figurative language. My focus will exclude idiomatic language and one-off analogies in favour of metaphor just because the latter is most interesting, though the methodology applies in other cases of figurative language. I will take a *metaphor* to be figurative language that utilises the systematic structure in one conceptual domain to give partial structure in another conceptual domain. This may subsume some cases of what would otherwise be called *similes* or *analogies*. The differences far as I can tell between a metaphor and an analogy is the presence of systematicity in the former, and a weak requirement that the correspondence involves separate conceptual domains. It doesn't really matter for this discussion what the difference is.

First, we observe that we can model certain kinds of analogies between conceptual spaces by considering structure-preserving maps between them. For example, Planck's law gives a partial continuous function from part of the positive reals measuring temperature of a black body in Kelvin to wavelengths of light emitted, and the restriction of this mapping to the visible spectrum gives the so-called "colour temperature" framework used by colourists. It will turn out that a decategorified cofunctor has the right kind of structure.

Second, we observe that we can use simple natural language to describe conceptual spaces, instead of geometric or topological models. Back to the example of colour temperature, instead of precise values in Kelvin, we may instead speak of landmark regions that represent both temperature and colour such as *incandescent* and *daylight*, which obey both temperature-relations (e.g. *incandescent* is cooler than *daylight* and colour-relations (e.g. *daylight* is bluer than *incandescent*).

Third, we observe that we can also use simple natural language to describe more complex conceptual schemes with interacting agents, roles, objects, and abilities. This will require a cofunctor. Organising this

[Red] argues convincingly that the metaphor IDEAS are CONTAINERS is pervasive in English; it is just about the only way we talk about communication. Yet there is no literal sense in which one can "get something" out of a lecture or "pack a lot" into a book. Evidently the systematicity of the metaphor itself yields the common structure from which we can even begin to consider pedestrian truth-conditional or possible-worlds analyses; i.e. language has a role to play in constructing the stage, and afterwards we can reason logically about the actors and events.

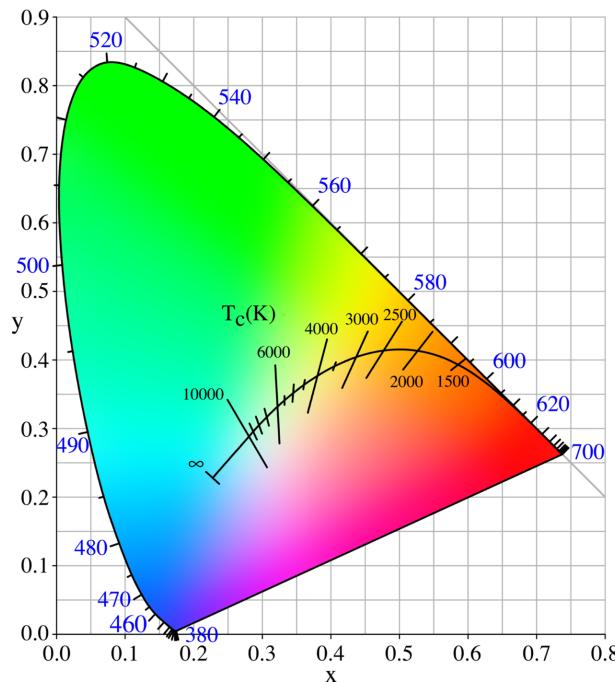
Meta, beyond. *Phor*, as in amphora, an agent, carrier, or producer. Metaphor carries meaning beyond one domain to another. It bears and produces meaning. Metaphors are the primary agents of meaning.

linguistic data in the concrete structure of a text circuit allows us to formally specify what it means for one conceptual scheme to structure another by describing structure-preserving maps between the text circuits. This will allow us construct topological models of metaphors such as **TIME is MONEY**.

5.6.1 Temperature and colour: the Planckian Locus

Example 5.6.1 (The Physicists' Planckian Locus). Planck's law describes the spectral radiation intensity of an idealised incandescent black body as a function of light frequency and temperature. Integrating over light frequencies in the visible spectrum yields a function from temperature of the black body to chromaticity.

Figure 5.11: The Planckian Locus in the CIE 1931 chromaticity diagram. Chromaticity refers only to the hue of a colour, without other domains such as saturation.



Abstractly, the Planckian Locus is a continuous function mapping the positive real line representing the conceptual domain of temperature into the plane representing the conceptual domain of colour. The Planckian locus is the basis of colourist-talk about colour schemes in terms of temperature, which allows them to coordinate movements in colourspace using the terminology of temperaturespace, e.g. **make this shot warmer**. This fits with what we would prototypically expect a metaphor to allow us to do with meanings.

However, the particular mathematical conception of metaphor-as-map in Example 5.6.1 is too rigid: it only goes one way. It is a specific and inflexible kind of metaphor that does not behave at all outside its specified

boundaries. For example, colourists have to deal with offsets towards green and magenta, which are not in the chromaticity codomain of the function given by Planck's law. It would be truer to life if we further analysed the function as mediated by a strip.

Example 5.6.2 (The colourist's Planckian Locus). Now we aim to extend our mathematical model to accommodate the fact that colourists deal with chromatic offsets or deviations from the mathematically precise locus given by Planck's law.

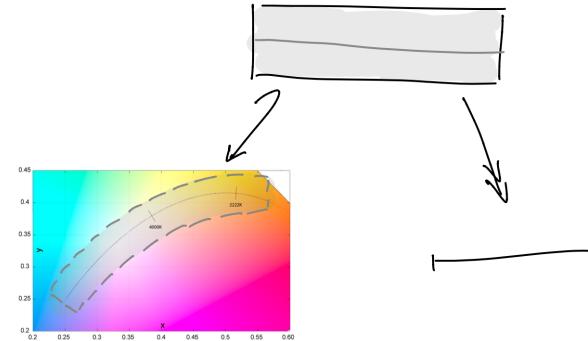


Figure 5.12: Consider the unit square (depicted as a strip) as a fiber bundle over the unit interval representing temperature range. There is an injective continuous map from the strip into colourspace that is centered on the Planck Locus.

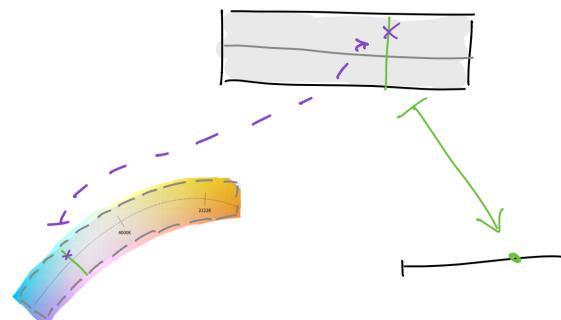


Figure 5.13: The left leg is bijective in the image restriction, so any point or displacement in the offset-strip in colourspace can be lifted to a point in the apex strip, which is then projected down along with other points in the vertical fiber to a point in temperaturespace. So we have a decategorified cofunctor!

A refinement we have just captured is the partially-structuring nature of metaphor [LJ03]. In the language of our running example, pure green is outside the scope of the colour-temperature correspondence given by the Planckian Locus, so the metaphor is only a partial structuring of the colour domain according to the temperature domain. This partiality in the colour domain means that it would have been inappropriate to model the passage of colour-talk to temperature-talk as a function from colour to temperature, as functions are total, rather than partial, on their domain. While it is conceptually nice that we are on the way to recovering

monoidal cofunctors as a model of metaphor, why didn't we stay simple and just use a partial function? The answer is that the strip at the apex represents the *talk* part of colour- and temperature-talk.

Example 5.6.3 (Conceptual transfer between domains). When colourists use the temperature metaphor they might say "hot", "warm", "cooler", which are not specific temperature ranges in Kelvin, but concepts in temperature-space. Recalling that we may consider concepts to be open sets of a topology (and comparatives as opens of the product), we observe that we can linguistically model regions on the positive reals with words **little** (labelled l), **lot** (labelled L), and **more** (labelled M), an algebraic basis from which derive **less** by symmetry, and other regions such as **more than a little**, **less than a lot**. In this particular running example, it happens that both legs of the span of functors have a lifting property, which explains how we might model the fact that conceptual colourist-talk of "daylight" or "candlelight" in the colour domain can be sensibly interpreted in the temperature domain. The formalisation of this fact follows by symmetry from this example.

Figure 5.14: Starting from the right, the lifting property of the right leg is what lets us map "hotter and colder" temperature talk into the more abstract quantity-talk of "more and less" in the apex strip. Then the left functor sends quantity-talk into the colour domain, which allows "hotter and colder" to be used in the colour domain.

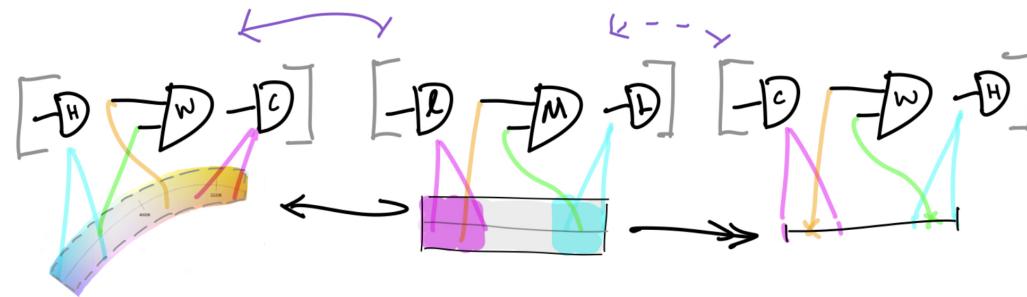
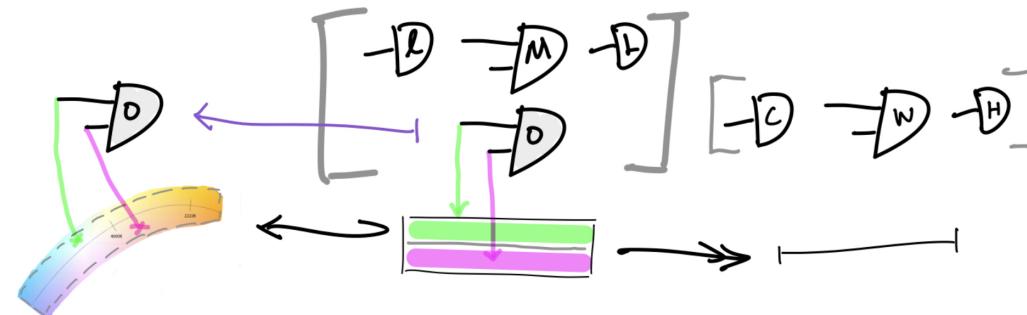


Figure 5.15: The additional expressive power that the apex strip gives is the concept of vertical offset, which doesn't appear in the real line. So the apex strip allows talk of quantity and offset, and this offset, when translated into the colour domain, allows talk of offset towards and away from, for instance, green.



5.6.2 Time and Money: complex conceptual structure

Metaphor is perhaps the only methodology we have for making sense of certain abstract concepts, such as Time. For example, many languages make use of the metaphor TIME is SPACE, in which space-talk is used to structure time. In English, the future is ahead of us and the past behind, while conversely, for the Aymara the future is behind and the past is ahead. Orthogonally, in Mandarin the future is below and the past is above. We have already demonstrated that we have the tools to deal with conceptual transfer between static conceptual spaces viewed as topological spaces via spans of continuous maps. What is of concern to us are *dynamic* metaphors that involve a conceptual space-time with agents and capabilities and so on. The following discussion draws heavily from [LJ03].

For example, in English, we make ample use of the metaphor TIME is MONEY. Now here are two mathematical observations about TIME is MONEY.

First, the conceptual affordances of money-talk are marshalled to give structure to time-talk, where there is no such structure were it not for the metaphor. To establish this first point of conceptual transfer by example, a phrase like *Do you have time to look at this?* is completely sensible to us, but literally meaningless; even if we had an oracle to measure possession, what would we point it at to measure a person's possession of time? Even if we accept some argument that the concept of possession is an innately human faculty, when we say *This is definitely worth your time!* or *What a waste of time.*, we are drawing upon value-talk that is properly contingent in the socially-constructed sense upon the conceptual complex of money.

Second, metaphor has a partial nature mediated by users and contexts, in that it is not the case that the metaphor licenses all kinds of money-talk to structure time-talk. To establish this second point of partiality, consider that money can be stored in a bank, whereas there is no real corresponding thing in the common conceptual vocabulary which one can store time and withdraw it for later use — Although, in a wonderful example of ‘pataphysical thinking, “Time Banks” have existed since the 19th century, which are practices of reciprocal service exchange that use units of time as currency. But the partiality constraint is itself partial. For instance, one can invest money into an enterprise in the expectation of greater returns, and this is not appropriate for many domains of time-talk, but there is a metaphorical match in some specific contexts, such as text-editor-talk: *learning vim slows you down at first but it will save you time later.*

Now I'll try to demonstrate by example that the kinda-cofunctors we explored in Section 2.1 between text circuits do all of the things we have asked for. The components of text circuits serve as an algebraic basis for dynamic conceptual complexes, while the kinda-cofunctor handles partial structuring of one conceptual domain in terms of another.

A preliminary observation is that the metaphor is a social construct, as we could just as well have collectively settled on a convention that TIME is FOOD, which provides a liberating sense of mastery (at least in a context where food is abundant): time can be prepared, produced, consumed, spiced if dull and best shared with loved ones.

Example 5.6.4 (Vincent spends his morning writing). To begin a formal figurative interpretation via the metaphor TIME is MONEY, we require some model of the conceptual domain of money, as well as a topological interpretation. As a first pass, we understand that money can be exchanged for goods and services, so we will settle for a text-circuit signature for trade to serve as the conceptual domain as the apex of a cofunctor, given in Figure 5.16. The elements of the topological model are given in Figure 5.17. The behaviour of the opfibration part of the cofunctor is detailed in Figure 5.18, and that of the identity-on-objects functor in Figures 5.19, 5.20, and 5.21. The figurative model serves as a foundation from which truth-theoretical semantics can begin. In the sketched interpretation, there aren't too many interesting questions one can ask, but the purpose of this example is to point out that in principle, we can exploit the systematicity of metaphor by constructing figurative mechanical models for which interesting questions can be asked and answered truth-theoretically, as in Figure 5.22.

Figure 5.16: In the TRADE signature, we define two roles as wires: TRADERS and TRADEABLES. There is one static relation HAS to indicate a trader's ownership of a tradeable, which can be further elaborated with equations to indicate e.g. exclusivity of ownership by interpreting violations of exclusivity as a zero morphism, assumed but elided for brevity. There is one dynamic verb (treated as a homotopy) TRADE, which at time 0 enforces a precondition that the traders have their respective tradables, and at time 1 (completion of the trade), the traders swap possession of their tradeables. The TRADE signature contains all nominal instantiations of nouns with respect to roles, which will be illustrated shortly.

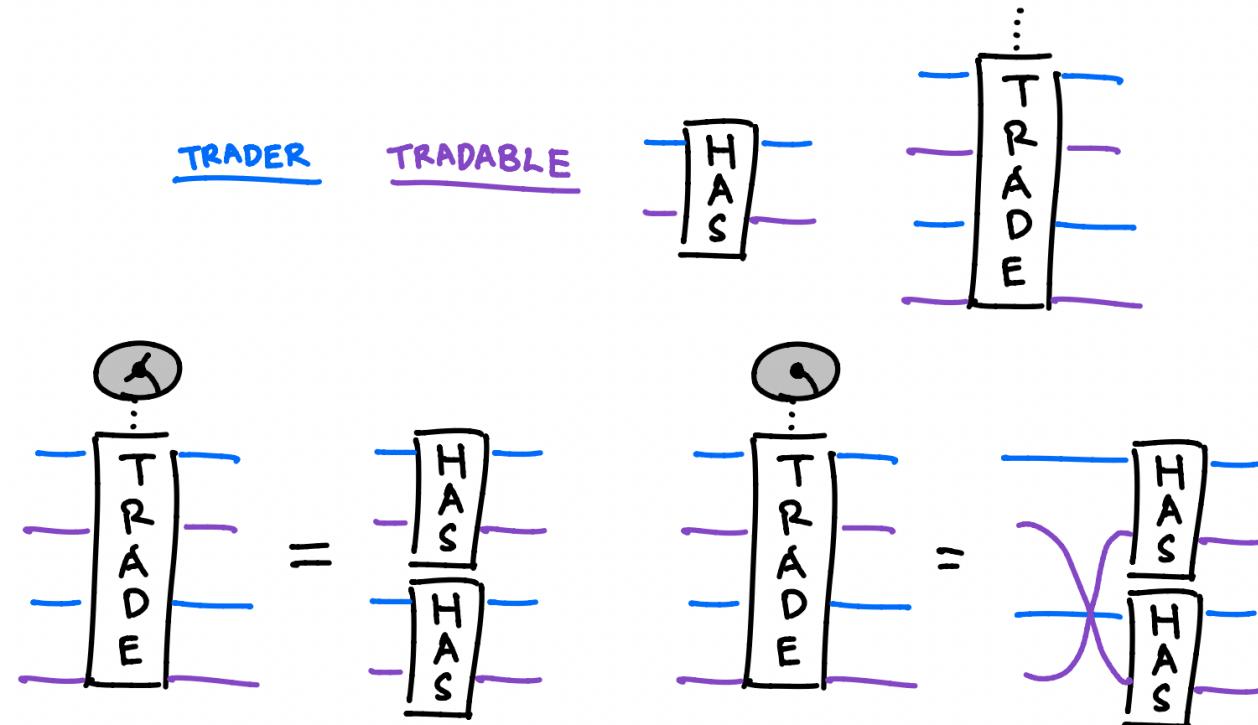


Figure 5.17: We build the topological model from two sticky spiders in the Euclidean plane. The TRADER spider will distinguish two regions of possession, so that HAS may be interpreted as a region-test. The TRADEABLES spider will specify four meeples or counters, three for time, and one for thesiswriting; we will use the configuration space of the TRADEABLES spider to regulate their movement and distribution. Next, we have to specify what the discrete opfibration is doing. Recalling our functor box notation, we can consider the job of the discrete opfibration to be role assignment from the verb SPEND in the utterance to the verb TRADE in the conceptual domain.

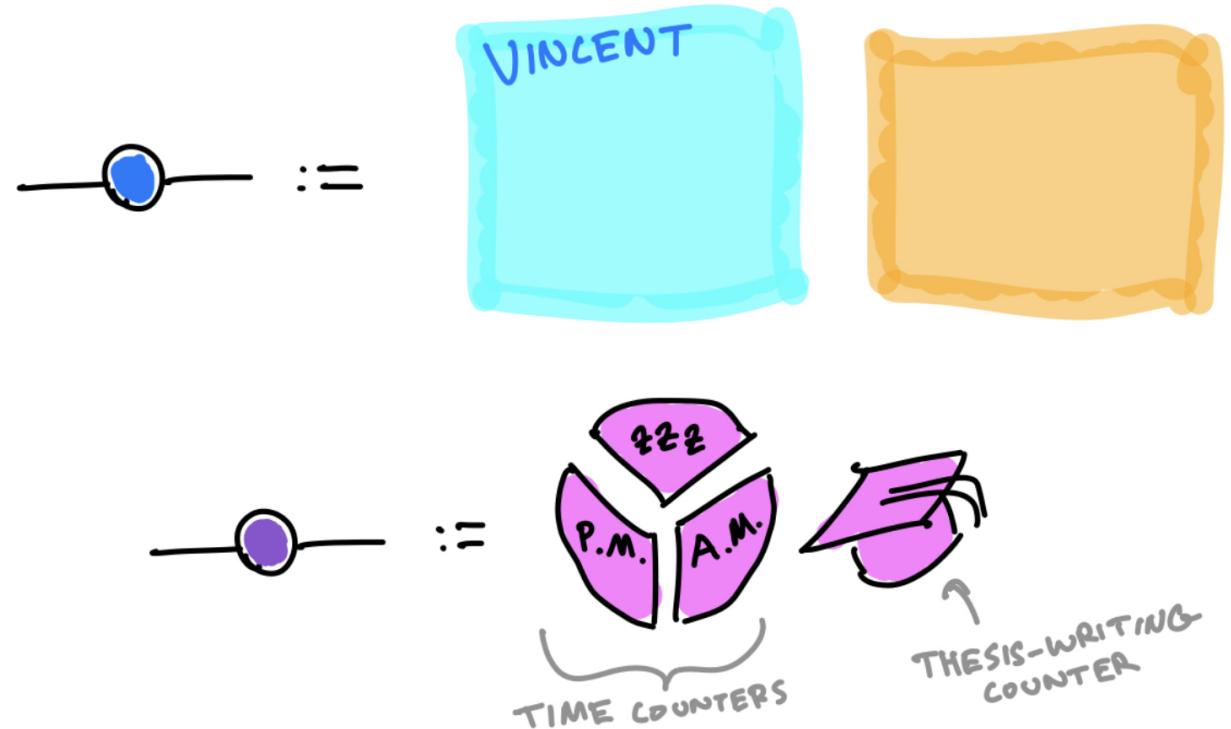


Figure 5.18: The opfibration forgets about role-assignments in its domain by sending them to the monoidal unit. The lift of the opfibration is a role-assignment. (Arguably) unambiguously in this example, Vincent is the spender and the first trader, and A.M. is the cost and the first tradeable. However, there are two options to resolve writing treated as a noun-phrase in the role of GOOD. In the first lift, writing is resolved as the other trader, and the implicit good as thesis. In the second lift, writing is the tradeable and something else is the trading counterparty, such as the world.

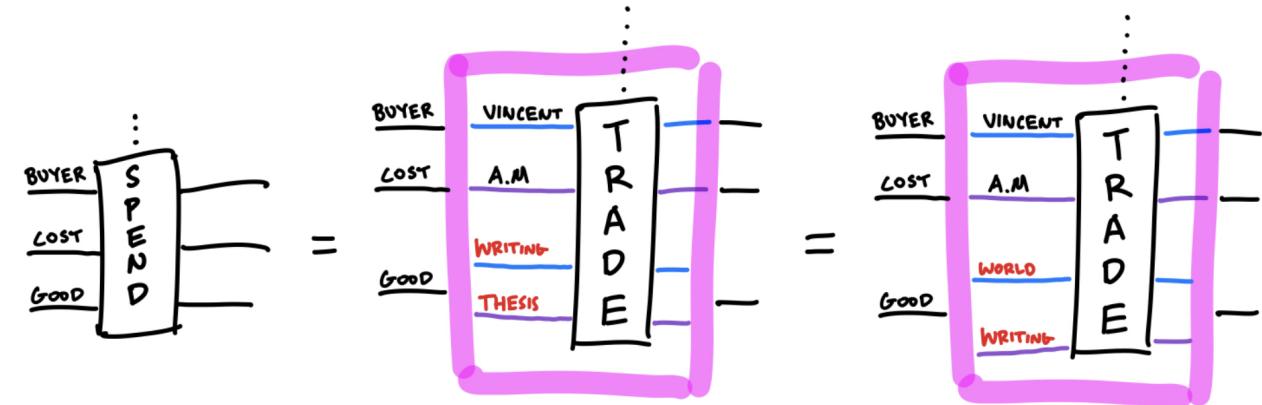


Figure 5.19: The section of the opfibration over the SPEND verb is a tabulation of all the ways in which conceptual roles in the TRADING domain can be assigned. To continue the example, we will assume the first lift in Example 5.18 as our interpretation. The identity-on-objects functor part of the cofunctor maps our chosen interpretation into the following diagram in **ContRel**. The configuration space of the TRADEABLES spider is expanded via split idempotent so that all thin wires in the diagram are typed as the Euclidean plane. Recalling Example 5.1.1, HAS is interpreted as the intersection of the position of a counter with the possessive region of the respective trader.

Figure 5.20: We may verify that the equations governing TRADE cohere with our topological figures. At time 0, before the trade, we can calculate that the permissible figures have Vincent in possession of A.M and writing in possession of thesis.

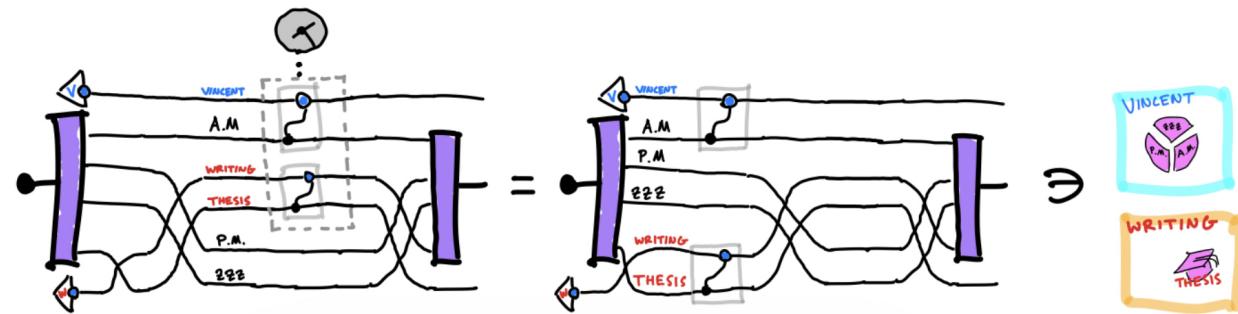
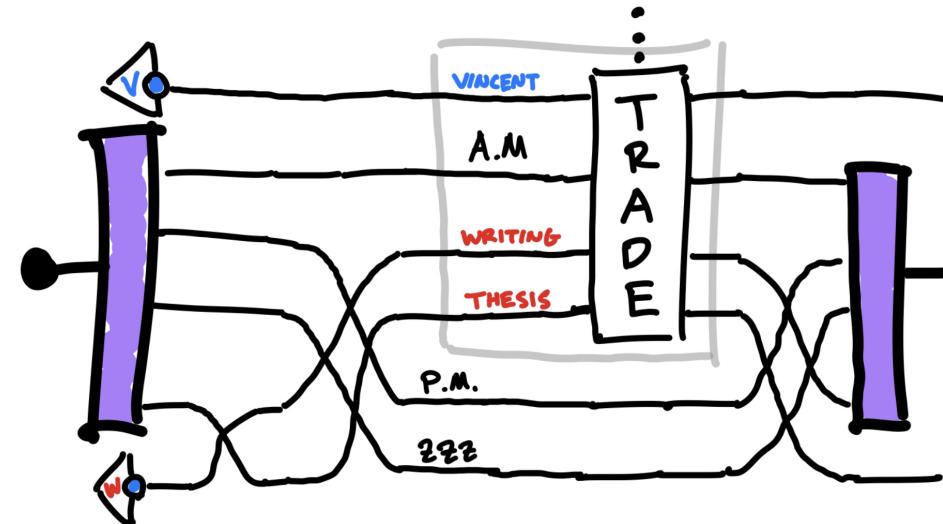


Figure 5.21: At time 1, we may calculate that the permissible figures must be such that Vincent is in possession of thesis and writing is in possession of what was previously my morning.

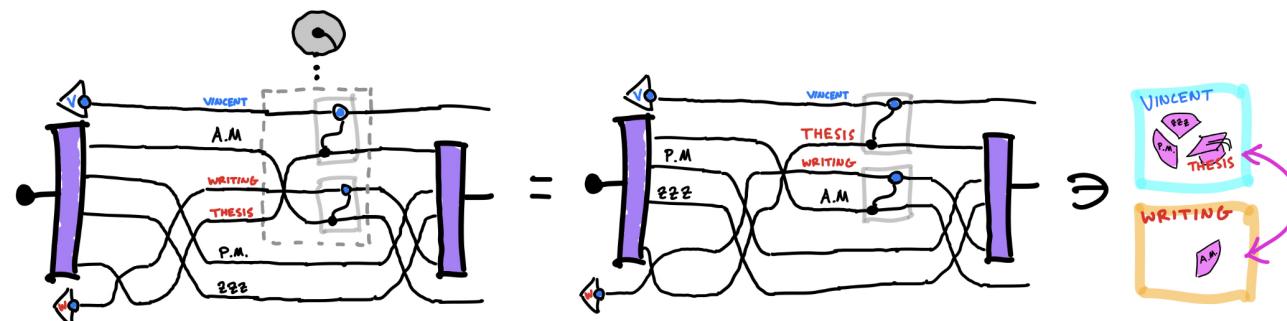
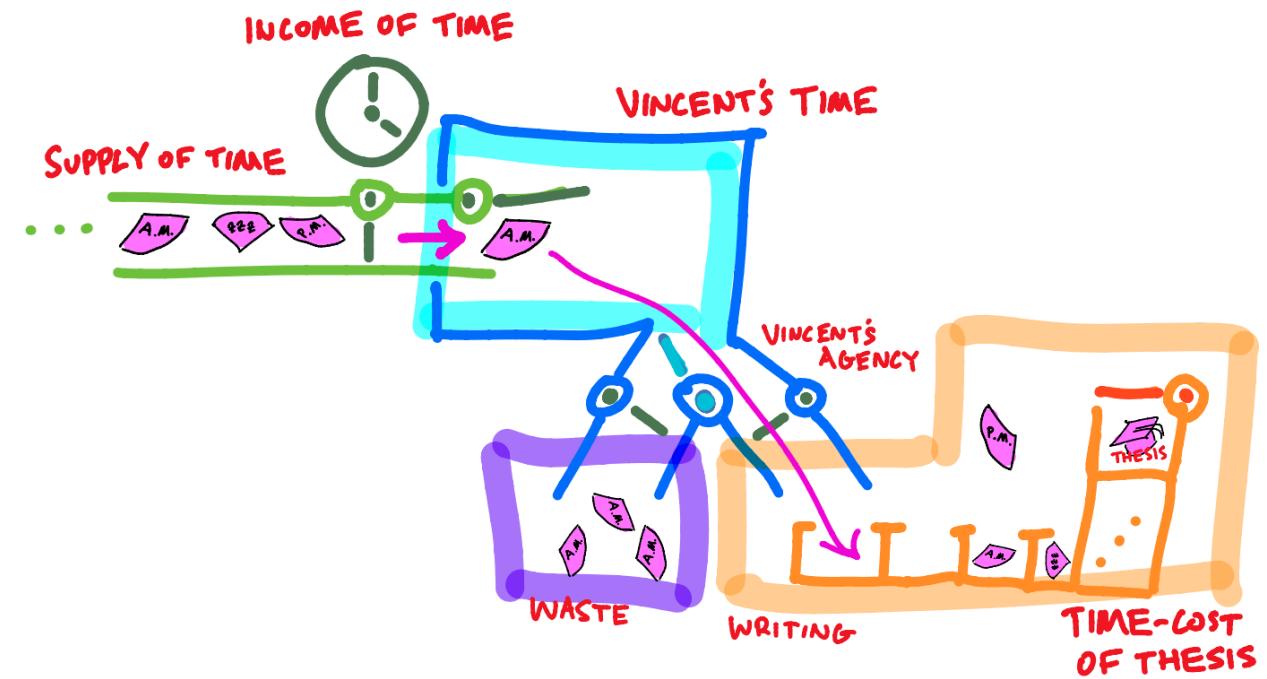


Figure 5.22: In a more detailed conceptual model of TIME is MONEY, rather than just TRADE, we might consider income, the spender's agency, and cost. In Euclidean 3-space, we might model income as a clock-gated mechanism that deposits time-tokens serially into Vincent's possession, along with his agency as a gated chute, and the time cost of writing a thesis as a dispenser that requires a certain number of tokens to release a thesis-token into Vincent's possession. In this sketch model, one obtains short films for He used to waste his mornings but now he spends them writing, or He once spent an evening writing but made no progress. One can then ascertain certain consequences truth-theoretically; for instance that there is at least one morning that was not spent on writing, or that there is at least one evening spent on writing but not inside the slot that would help a thesis-release mechanism trigger. In every case, cofunctionality handles bookkeeping for role-interpretation choices and guarantees systematicity of the topological figure according to the signature at the apex model that models the organising concept.



5.7 A specimen problem sheet from an imaginary future

In multiple reveries while writing, I audited a class on string-diagrammatic methods in formal linguistics someplace in an unlikely future. The people there had strange views on facets of language as a formal object, which I'll try to summarise here.

Composition Compositionality equals topological representability; in particular, meaning relations in text are witnessed by connectivity of string diagrams.

Systematicity Systematicity equals functorially witnessed relations between compositional structures; in particular, spans of functors between families of string diagrams witness agreement between different theories as topological equivalence.

Syntax Syntax equals a coherent method of synthesising *and* analysing composition; in particular, any internally consistent conception of natural language syntax in terms of string diagrams is permissible.

Semantics Semantics equals computation; in particular, any consistent computational interpretation of the content of string diagrams is permissible.

These beliefs led to a wild form of picturalism. I tried to explain that there was an essential tension between the detail of rigour and the creative-expressive aspect of formal models, but their synthesis of this dichotomy was so ingrained they could not understand what the problem was. It was pretty liberating stuff. With their permission, I took home a suitably redacted problem sheet and filled it in myself, which I share with you now.

Postscript: That was the summary of the thesis. Thanks and goodbye.

Prof. [REDACTED]

[REDACTED]: Diagrammatic Methods in [REDACTED]

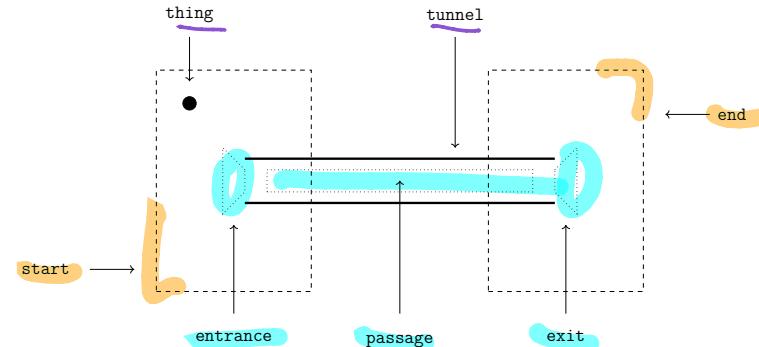
Date: [REDACTED], 20[REDACTED]

Problem Sheet [REDACTED] (due Week [REDACTED])

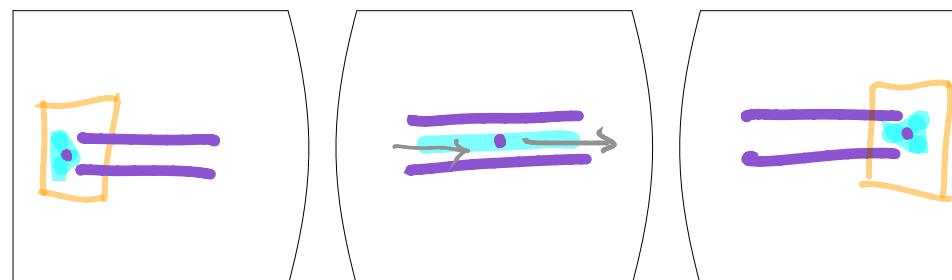
Question 1.

You may assume standard semantics of motion.

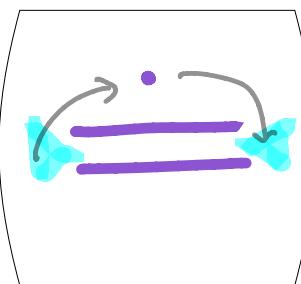
Consider the following iconic signature for the TUNNEL concept, with two movable shapes (in bold) and two place-indexings (indicated by dashed and dotted lines.)



a) Fill the 3-panel vignette in the TUNNEL signature for thing goes through the tunnel.



b) Depict an intermediate panel such that its splice is not a 3-panel vignette for the same sentence.



c) Briefly justify your answer for part b).

Through \Rightarrow Across ($\text{start} \rightarrow \text{end}$), but converse doesn't hold.

d) Hence, or otherwise, provide process typing and relations for through in TUNNEL.

Assuming through as dynamic in config space of $\{\text{thing}, \text{tunnel}\}$



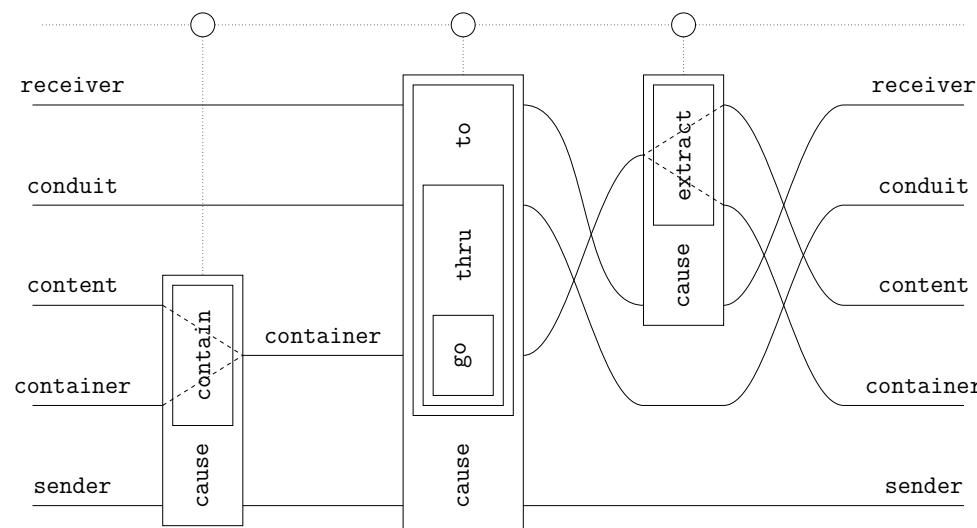
Assume $\text{ent} \in \text{start}$, i.e. $\text{N} \circ = \text{N}$ [symmetrically for $\text{exit} \in \text{end}$]

$$\begin{array}{c} \text{N} \\ \text{---} \\ \text{N} \end{array} = \begin{array}{c} \text{N} \\ \text{---} \\ \text{N} \end{array} = \text{N} \quad \left[\begin{array}{l} \text{typing by config-space of rigid mot. in plane.} \\ \text{"At time } 0 \text{ thing is at entrance"} \\ \text{"At time } 1 \text{ thing is at exit (by sym.)"} \end{array} \right]$$

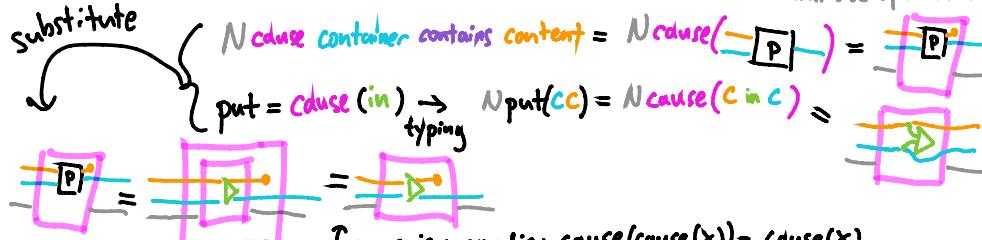
$$\begin{array}{c} \text{N} \\ \text{---} \\ \text{N} \end{array} = \begin{array}{c} \text{N} \\ \text{---} \\ \text{N} \end{array} = \text{N} \quad \left[\begin{array}{l} \text{"At time } < 1 \text{ thing is either at ent. or in passage"} \\ \text{This rules out counter. in part b) of Q.} \end{array} \right]$$

Question 2.

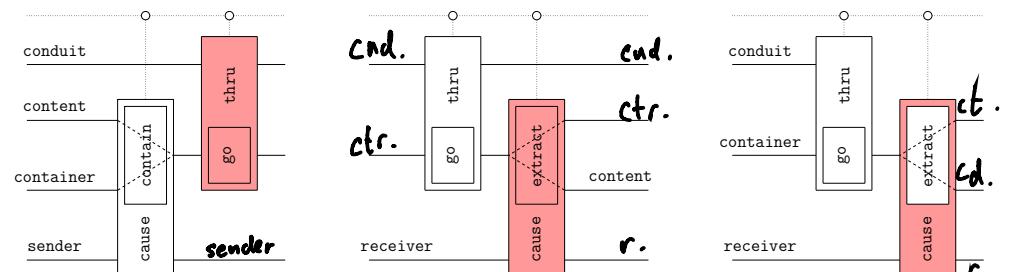
Consider the following to be the characteristic process of the CONDUIT metaphor.



- a) Analyse and equationally characterise (N)cause(contains) in terms of the dynamic verb put and static relation in. You may assume put = cause(in) and standard put-get typing. ↳ will use update-struct.



- b) Assuming standard negation semantics, label and gloss the following composites.



put in but not go through.
Container
*gone through but
(could/did) not extract
from container.*

*container goes through.
contents extracted but
not by receiver
(e.g. falls out)*

- c) For each composite, give an example of a matching sentence in the CONDUIT metaphor, along with brief justification.

1. The painstaking speech fell on deaf ears.

J: painstaking speech \rightarrow ideas put in words deaf ears \Rightarrow words did not go through.

2. They heard him, but they didn't listen.

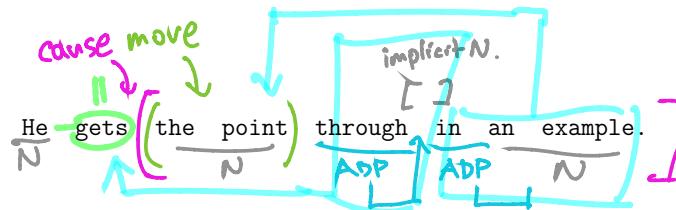
J: heard him \rightarrow words came through didn't listen \rightarrow ideas/meaning not extracted from words.

3. His message was crystal-clear.

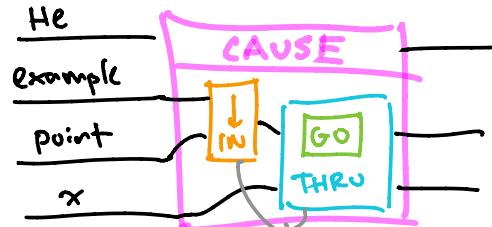
J: crystal-clear \equiv transparent \rightarrow contents extracted without effort
(visible)

Question 3.

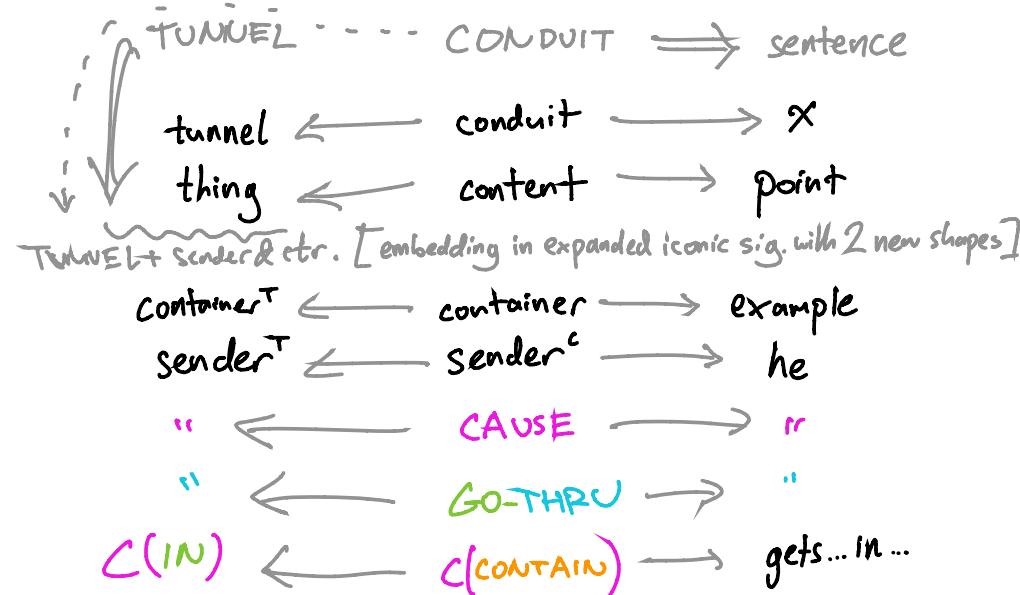
- a) Provide a phrasal analysis. You may ignore contextual determiners.



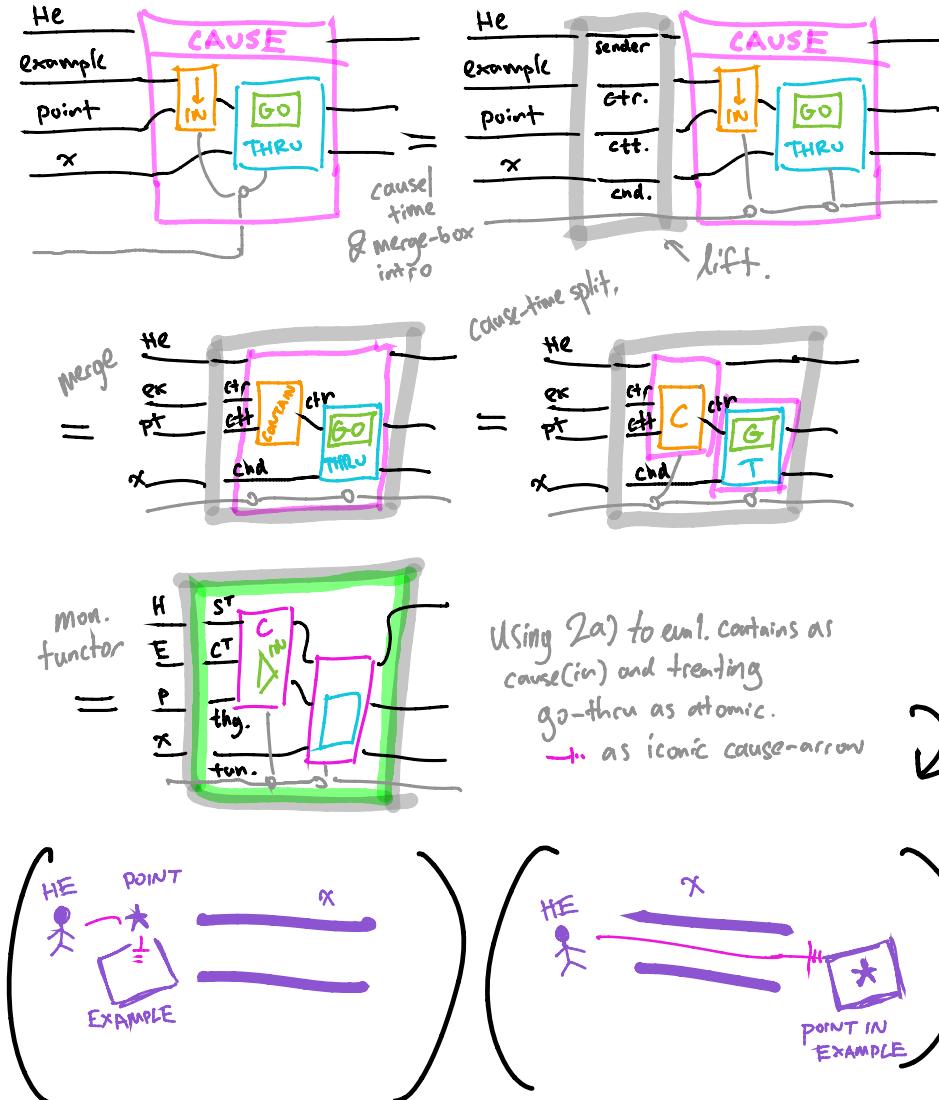
- b) Hence, or otherwise, provide a text circuit for the sentence.



- c) Using your answers from previous questions, compute the iconic semantics in TUNNEL by merge-boxes. You may assume standard causal semantics and notation. Justify nonstandard notations.



(An additional blank page is provided for calculation, should you need it.)



Notices: Due to an ongoing [REDACTED] weather-event there is a [REDACTED] infestation in Hall [REDACTED]. Next week's lectures will take place on [REDACTED]; a reminder that non-[REDACTED] students must have their [REDACTED]-modules [REDACTED] and [REDACTED] AT ALL TIMES WITHOUT EXCEPTION.

6

Bibliography

- [A q] A quantum computer. Ludovico Quanthoven.
- [Abr09] Samson Abramsky. Abstract Scalars, Loops, and Free Traced and Strongly Compact Closed Categories, October 2009. Comment: 32 pages.
- [ano] anonymous Quantinuum researchers. DisCoCirc: Compositional Discourse Structure for Interpretable Machine Learning.
- [BAB] bAbI - Meta Research. <https://research.facebook.com/downloads/babi/>.
- [Bae97] John C. Baez. An Introduction to n-Categories, May 1997. Comment: 34 pages LaTeX, 30 encapsulated Postscript figures, 2 style files.
- [Bar] Michael Barr. NOTE ON A THEOREM OF PUTNAM'S. *Theory and Applications of Categories*, 3(3).
- [Bas22] Matthias Bastian. Google PaLM: Giant language AI can explain jokes. <https://the-decoder.com/google-palm-giant-language-ai-can-explain-jokes/>, April 2022.
- [BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. Comment: 156 pages. Work in progress – comments welcome!
- [BCG⁺17] Joe Bolt, Bob Coecke, Fabrizio Genovese, Martha Lewis, Dan Marsden, and Robin Piedeleu. Interacting Conceptual Spaces I : Grammatical Composition of Concepts. <https://arxiv.org/abs/1703.08314v2>, March 2017.
- [BDGHS24] Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Paweł Sobociński. Diagrammatic Algebra of First Order Logic, January 2024.

- [Ben64] J. Benabou. Algèbre élémentaire dans les catégories avec multiplication. *C. R. Acad. Sci., Paris*, 258:771–774, 1964.
- [BK20] Emily M. Bender and Alexander Koller. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online, 2020. Association for Computational Linguistics.
- [BM] Wojciech Buszkowski and Katarzyna Moroz. Pregroup Grammars and Context-free Grammars.
- [BM20] John C. Baez and Jade Master. Open Petri Nets. *Math. Struct. Comp. Sci.*, 30(3):314–341, March 2020. Comment: 30 pages, TikZ figures.
- [Boh09] Jürgen Bohnemeyer. Temporal anaphora in a tenseless language. In Wolfgang Klein and Ping Li, editors, *The Expression of Time*, pages 83–128. Mouton de Gruyter, March 2009.
- [BPSZ19] Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Graphical Affine Algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2019.
- [BS22] Guillaume Boisseau and Paweł Sobociński. String Diagrammatic Electrical Circuit Theory. *Electron. Proc. Theor. Comput. Sci.*, 372:178–191, November 2022. Comment: In Proceedings ACT 2021, arXiv:2211.01102.
- [BSC15] Esma Balkir, Mehrnoosh Sadrzadeh, and Bob Coecke. Distributional Sentence Entailment Using Density Matrices, October 2015. Comment: 11 pages.
- [BSZ14] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A Categorical Semantics of Signal Flow Graphs. In *CONCUR 2014 - Concurrency Theory - 25th International Conference*, volume CONCUR 2014 - Concurrency Theory - 25th International Conference, September 2014.
- [BSZ17] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Interacting Hopf Algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, January 2017.
- [Car88] Rudolf Carnap. *Meaning and Necessity: A Study in Semantics and Modal Logic*. University of Chicago Press, Chicago, IL, February 1988.
- [CD11] Bob Coecke and Ross Duncan. Interacting Quantum Observables: Categorical Algebra and Diagrammatics. *New J. Phys.*, 13(4):043016, April 2011.
- [CDH20] Cole Comfort, Antonin Delpeuch, and Jules Hedges. Sheet diagrams for bimonoidal categories, December 2020.

- [CE11] Bob Coecke and Bill Edwards. Three qubit entanglement within graphical Z/X-calculus. *Electron. Proc. Theor. Comput. Sci.*, 52:22–33, March 2011.
- [CFH⁺09] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In Richard F. Paige, editor, *Theory and Practice of Model Transformations*, Lecture Notes in Computer Science, pages 260–283, Berlin, Heidelberg, 2009. Springer.
- [CG16] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016.
- [CG23] Bob Coecke and Stefano Gogioso. *Quantum in Pictures: A New Way to Understand the Quantum World*. Cambridge Quantum, February 2023.
- [CGG⁺22] Geoffrey SH Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *Programming Languages and Systems: 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*, pages 1–28. Springer International Publishing Cham, 2022.
- [CH08] J. R. B. Cockett and P. J. W. Hofstra. Introduction to Turing categories. *Annals of Pure and Applied Logic*, 156(2):183–209, December 2008.
- [Chapm] Chapman, David. Nebulosity | Meaningness. <https://meaningness.com/nebulosity>, Dec. 15, 2010, 10:33 p.m.
- [Cho75] Man-Duen Choi. Completely positive linear maps on complex matrices. *Linear Algebra and its Applications*, 10(3):285–290, June 1975.
- [Cho00] Noam Chomsky. *New Horizons in the Study of Language and Mind*. Cambridge University Press, Cambridge, 2000.
- [Chu33] Alonzo Church. A Set of Postulates For the Foundation of Logic. *Annals of Mathematics*, 34(4):839–864, 1933.
- [Chu11] Kenneth Church. A Pendulum Swung Too Far. *LiLT*, 6, October 2011.
- [CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017.

- [CKWW07] A. Carboni, G. M. Kelly, R. F. C. Walters, and R. J. Wood. Cartesian bicategories. II. *Theory and Applications of Categories [electronic only]*, 19:93–124, 2007.
- [Cla23] Bryce Clarke. *The Double Category of Lenses*. Thesis, Macquarie University, February 2023.
- [CND⁺22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellar, 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.
- [Coe04] Bob Coecke. The logic of entanglement, March 2004.
- [Coe20] Bob Coecke. The Mathematics of Text Structure, February 2020.
- [Coe21] Bob Coecke. Compositionality as we see it, everywhere around us, October 2021.
- [CPP09] Bob Coecke, Eric Oliver Paquette, and Dusko Pavlovic. Classical and quantum structuralism, April 2009.
- [CPV13] Bob Coecke, Dusko Pavlovic, and Jamie Vicary. A new description of orthogonal bases. *Math. Struct. Comp. Sci.*, 23(3):555–567, June 2013.
- [Cro95] William Croft. Autonomy and Functionalist Linguistics. *Language*, 71(3):490–532, 1995.
- [CSC10] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning, March 2010. Comment: to appear.
- [CW87] A. Carboni and R. F. C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49(1):11–32, November 1987.
- [CW21] Bob Coecke and Vincent Wang. Grammar Equations, June 2021.
- [dav] davidad. An Open Agency Architecture for Safe Transformative AI.

[Del20] Antonin Delpeuch. Autonomization of Monoidal Categories. *Electron. Proc. Theor. Comput. Sci.*, 323:24–43, September 2020.

[Dis] displaCy Dependency Visualizer · Explosion. <https://explosion.ai/displacy>.

[DLS⁺23] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality, June 2023.

[Dor23] Christoph Dorn. Associative \$n\$-categories, March 2023.

[DV22] Andrew Dudzik and Petar Veličković. Graph Neural Networks are Dynamic Programmers, October 2022.

[dZ14] Christian Schröder de Witt and Vladimir Zamdzhev. The ZX-calculus is incomplete for quantum mechanics. *Electron. Proc. Theor. Comput. Sci.*, 172:285–292, December 2014.

[Eve09] Daniel L. Everett. *Don't Sleep, There Are Snakes: Life and Language in the Amazonian Jungle*. Vintage, New York, illustrated edition edition, November 2009.

[FB01] C. Fillmore and Collin F. Baker. Frame semantics for text understanding. 2001. [TLDR] An introduction to knowledge representation using Frame Semantics, as is being carried out in the FrameNet Project, providing examples of many of the questions being dealt with and the proposed solutions, including semantic composition, text coherence, polysemy and WSD, and evidentiality.

[FGP21] Tobias Fritz, Tomáš Gonda, and Paolo Perrone. De Finetti's Theorem in Categorical Probability. *josa*, 2(4), November 2021.

[FHC05] W. Tecumseh Fitch, Marc D. Hauser, and Noam Chomsky. The evolution of the language faculty: Clarifications and implications. *Cognition*, 97(2):179–210; discussion 211–225, September 2005.

[Fir57] John Rupert Firth. *Studies in Linguistic Analysis*. Publications of the Philological Society. Blackwell, Oxford, 1957.

[Flo14] Luciano Floridi. *The Fourth Revolution: How the Infosphere Is Reshaping Human Reality*. OUP Oxford, New York ; Oxford, June 2014.

[FP88] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

- [Fre84] Frege, Gottlob. Selbst concrete Dinge sind nicht immer vorstellbar. Man muss die Wörter im Satze betrachten, wenn man nach ihrer Bedeutung fragt. In *Die Grundlagen Der Arithmetik*. 1884.
- [Fri05] Harvey Friedman. [FOM] Characterization of R/Simple proof, Tue Feb 15 02:13:25 EST 2005.
- [FS18] Brendan Fong and David I. Spivak. Hypergraph Categories.
<https://arxiv.org/abs/1806.08304v3>, June 2018.
- [Gär14] Peter Gärdenfors. *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. January 2014.
- [Gib12] Jeremy Gibbons. Relating Algebraic and Coalgebraic Descriptions of Lenses. *Electronic Communications of the EASST*, 49 (Bidirectional Transformations 2012), 2012.
- [GPM⁺14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014.
- [Had17] A. Hadzihasanovic. *The Algebra of Entanglement and the Geometry of Composition*.
<http://purl.org/dc/dcmitype/Text>, University of Oxford, 2017.
- [Har93] Randy Allen Harris. *The Linguistics Wars*. Oxford University Press USA, New York, NY, reprint edition edition, January 1993.
- [har19] Lecture 1A: Overview and Introduction to Lisp, August 2019.
- [HBK⁺21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset, November 2021. Comment: NeurIPS 2021. Code and the MATH dataset is available at <https://github.com/hendrycks/math/>.
- [Hed15] Jules Hedges. String diagrams for game theory, March 2015.
- [Her12] Suzana Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proc Natl Acad Sci U S A*, 109 Suppl 1(Suppl 1):10661–10668, June 2012.
- [HK98] Irene Heim and Angelika Kratzer. *Semantics in Generative Grammar*. Malden, MA: Blackwell, 1998.
- [HRV22] Lukas Heidemann, David Reutter, and Jamie Vicary. Zigzag normalisation for associative \$n\$-categories. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, August 2022.

- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [HS20] Nathan Haydon and Paweł Sobociński. Compositional Diagrammatic First-Order Logic. In Ahti-Veikko Pietarinen, Peter Chapman, Leonie Bosveld-de Smet, Valeria Giardino, James Corder, and Sven Linker, editors, *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science, pages 402–418, Cham, 2020. Springer International Publishing.
- [Hu] Nick Hu. External traced monoidal categories.
- [HWW20] James Hefford, Vincent Wang, and Matthew Wilson. Categories of Semantic Concepts, August 2020. Comment: Accepted at SemSpace 2020.
- [Inq] Inquisitive Semantics - Inquisitive Semantics. <https://projects illc.uva.nl/inquisitivesemantics/>.
- [Jam72] A. Jamiołkowski. Linear transformations which preserve trace and positive semidefiniteness of operators. *Reports on Mathematical Physics*, 3(4):275–278, December 1972.
- [Jea67] Piaget Jean. *Childs Conception Of Space*. W. W. Norton and Company, Inc., May 1967.
- [Jef98] Alan Jeffrey. Premonoidal categories and flow graphs. *Electronic Notes in Theoretical Computer Science*, 10:51, January 1998.
- [JKZ19] Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal Inference by String Diagram Surgery, July 2019.
- [Jos87] Aravind K. Joshi. An Introduction to Tree Adjoining Grammar. In Alexis Manaster-Ramer, editor, *Mathematics of Language: Proceedings of a Conference Held at the University of Michigan, Ann Arbor, October 1984*, page 87. John Benjamins Publishing Company, January 1987.
- [Joy] André Joyal. THE GEOMETRY OF TENSOR CALCULUS II.
- [JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991.
- [JZ21] Theo M. V. Janssen and Thomas Ede Zimmermann. Montague Semantics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2021 edition, 2021.
- [Kan19] Pentti Kanerva. Computing with High-Dimensional Vectors. *IEEE Des. Test*, 36(3):7–14, June 2019.
- [kar23] Lambeq. Cambridge Quantum, June 2023.

- [Kha23] Tabarak Khan. What are tokens and how to count them?, 2023.
- [KLLW24] Nikhil Khatri, Tuomas Laakkonen, Jonathon Liu, and Vincent Wang-Maścianica. On the Anatomy of Attention, July 2024. Comment: Replaced to fix typos.
- [KPS14] Sebastian Kerkhoff, Reinhard Pöschel, and Friedrich Martin Schneider. A Short Introduction to Clones. *Electronic Notes in Theoretical Computer Science*, 303:107–120, March 2014.
- [KS16] Nikolaus Kriegeskorte and Katherine R. Storrs. Grid Cells for Conceptual Spaces? *Neuron*, 92(2):280–284, October 2016.
- [KW23] Philipp Koralus and Vincent Wang-Maścianica. Humans in Humans Out: On GPT Converging Toward Common Sense in both Success and Failure, March 2023.
- [Lak68] George Lakoff. Instrumental Adverbs and the Concept of Deep Structure. *Foundations of Language*, 4(1):4–29, 1968.
- [Lam58] Joachim Lambek. The Mathematics of Sentence Structure. *The American Mathematical Monthly*, 65(3):154–170, March 1958.
- [Lam61] Joachim Lambek. On the calculus of syntactic types. In Roman Jakobson, editor, *Proceedings of Symposia in Applied Mathematics*, volume 12, pages 166–178, Providence, Rhode Island, 1961. American Mathematical Society.
- [Lan10] Saunders Mac Lane. *Categories for the Working Mathematician: 5*. Springer, New York, NY, 2nd ed. 1978. softcover reprint of the original 2nd ed. 1978 edition edition, November 2010.
- [LAS21] Bastien Liétard, Mostafa Abdou, and Anders Søgaard. Do Language Models Know the Way to Rome? In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 510–517, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.
- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.
- [LGT23] Ziming Liu, Eric Gan, and Max Tegmark. Seeing is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability, May 2023.
- [lin18] The Neuroscience of Language and Thought, Dr. George Lakoff Professor of Linguistics, October 2018.

- [Liu21] Jonathon Liu. Language as Circuits. Master's thesis, The University of Oxford, Oxford, United Kingdom, 2021.
- [LJ03] George Lakoff and Mark Johnson. *Metaphors We Live By*. University of Chicago Press, Chicago, new edition edition, April 2003.
- [LPM⁺23] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. QNLP in Practice: Running Compositional Models of Meaning on a Quantum Computer. *Journal of Artificial Intelligence Research*, 2023.
- [LT23] Robin Lorenz and Sean Tull. Causal models in string diagrams, April 2023.
- [Mac63] Saunders MacLane. Natural Associativity and Commutativity. *Rice Institute Pamphlet - Rice University Studies*, 49(4), October 1963.
- [Mar77] D. Marr. Artificial intelligence—A personal view. *Artificial Intelligence*, 9(1):37–48, August 1977.
- [Mar10] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. MIT Press, July 2010.
- [Mel06] Paul-André Melliès. Functorial Boxes in String Diagrams. In Zoltán Ésik, editor, *Computer Science Logic*, Lecture Notes in Computer Science, pages 1–30, Berlin, Heidelberg, 2006. Springer.
- [Mer14] Alexander Merry. *Reasoning with !-Graphs*. PhD thesis, arXiv, March 2014. Comment: DPhil (PhD) thesis; University of Oxford; 172 pages.
- [MG17] Dan Marsden and Fabrizio Genovese. Custom Hypergraph Categories via Generalized Relations. <https://arxiv.org/abs/1703.01204v1>, March 2017.
- [ML20] Francois Meyer and Martha Lewis. Modelling Lexical Ambiguity with Density Matrices. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 276–290, Online, November 2020. Association for Computational Linguistics.
- [MN21] Marjorie McShane and Sergei Nirenburg. *Linguistics for the Age of AI*. March 2021.
- [Mon70] Richard Montague. Universal Grammar. *Theoria*, 36(3):373–398, 1970.
- [Mon73] Richard Montague. The Proper Treatment of Quantification in Ordinary English. In Patrick Suppes, Julius Moravcsik, and Jaakko Hintikka, editors, *Approaches to Natural Language*, pages 221–242. Dordrecht, 1973.
- [Moo14] Michael Moortgat. Typological Grammar. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2014 edition, 2014.

- [MP19] Francis Mollica and Steven T. Piantadosi. Humans store about 1.5 megabytes of information during language acquisition. *R Soc Open Sci*, 6(3):181393, March 2019.
- [Nat19] NativLang. Maya Before, Maya After: How a Tenseless Language Talks Past and Future, June 2019.
- [NBvV22] Rick Nouwen, Adrian Brasoveanu, Jan van Eijck, and Albert Visser. Dynamic Semantics. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2022 edition, 2022.
- [NC22] Sharan Narang and Aakanksha Chowdhery. Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance. <https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html>, 2022.
- [Ng18] Kang Feng Ng. *Completeness of the ZW and ZX Calculi*. PhD thesis, University of Oxford, 2018.
- [nLaa] nLab authors. Homotopy.io in nLab. <https://ncatlab.org/nlab/show/homotopy.io>.
- [nLab] nLab authors. Loc in nLab. <https://ncatlab.org/nlab/show/Loc>.
- [nLac] nLab authors. PROP in nLab. <https://ncatlab.org/nlab/show/PROP>.
- [nLad] nLab authors. Stabilization hypothesis in nLab. <https://ncatlab.org/nlab/show/stabilization+hypothesis>.
- [Ope22] OpenAI. ChatGPT: Optimizing Language Models for Dialogue. <https://openai.com/blog/chatgpt/>, November 2022.
- [Paq08] Éric Oliver Paquette. *Categorical Quantum Computation*. PhD thesis, Université de Montréal, 2008.
- [Par14] B. Partee. A Brief History of the Syntax-Semantics Interface in Western Formal Linguistics. 2014. [TLDR] This essay describes and comments on some of the key developments in the history of formal semantics and its relation to syntax, focusing on the period from the beginnings of Chomskyan generative grammar in the 1950's up until the mature period of formal language semantics in the 1980's.
- [Pav12] Dusko Pavlovic. Monoidal computer I: Basic computability by string diagrams, December 2012.
- [Pav14] Dusko Pavlovic. Monoidal computer II: Normal complexity by string diagrams, February 2014.
- [Pav23] Dusko Pavlovic. Programs as Diagrams: From Categorical Computability to Computable Categories, March 2023.

- [PH97] Barbara H. Partee and Herman L. W. Hendriks. Chapter 1 - Montague Grammar. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 5–91. North-Holland, Amsterdam, January 1997.
- [PP08] Paul H. Portner and Barbara H. Partee. *Formal Semantics: The Essential Readings*. John Wiley & Sons, April 2008.
- [PR69] P. Stanley Peters and Robert W. Ritchie. A note on the universal base hypothesis. *Journal of Linguistics*, 5(1):150–152, April 1969.
- [Put81] Hilary Putnam. *A Problem About Reference*. Cambridge University Press, 1981.
- [PWS⁺23] Boldizsár Poór, Quanlong Wang, Razin A. Shaikh, Lia Yeh, Richie Yeung, and Bob Coecke. Completeness for arbitrary finite dimensions of ZXW-calculus, a unifying calculus, April 2023.
- [PY18] Dusko Pavlovic and Muzamil Yahia. Monoidal computer III: A coalgebraic view of computability and complexity, March 2018.
- [Qui15] David Quick. *!-Logic*. PhD thesis, The University of Oxford, 2015.
- [Red] Michael J. Reddy. The conduit metaphor: A case of frame conflict in our language about language. *Metaphor and Thought*, page 164.
- [RFL⁺24] Benjamin Rodatz, Ian Fan, Tuomas Laakkonen, Neil John Ortega, Thomas Hoffman, and Vincent Wang-Mascianica. A Pattern Language for Machine Learning Tasks, July 2024.
- [Ric15] Richard Ridel. Mechanical Turing Machine in Wood, October 2015.
- [Ril22] Riley Goodside (@goodside) / Twitter. <https://twitter.com/goodside>, December 2022.
- [RM87] David E. Rumelhart and James L. McClelland. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MIT Press, 1987.
- [Rom21] Mario Román. Open Diagrams via Coend Calculus. *Electron. Proc. Theor. Comput. Sci.*, 333:65–78, February 2021.
- [RS24] Mario Román and Paweł Sobociński. String Diagrams for Premonoidal Categories, March 2024.
- [RV19] David Reutter and Jamie Vicary. High-level methods for homotopy construction in associative $\$n\$$ -categories, February 2019.

- [SCC13] Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. The Frobenius anatomy of word meanings I: Subject and object relative pronouns. *Journal of Logic and Computation*, 23(6):1293–1317, December 2013.
- [SCC16] Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. The Frobenius anatomy of word meanings II: Possessive relative pronouns. *J Logic Computation*, 26(2):785–815, April 2016. Comment: 40 pages, Journal of Logic and Computation, Essays dedicated to Roy Dyckhoff on the occasion of his retirement, S. Graham-Lengrand and D. Galmiche (eds.), 2014.
- [Sea80] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–424, September 1980.
- [Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. volume 813, pages 289–355. 2010.
- [SH05] Bernard Suits and Thomas Hurka. *The Grasshopper: Games, Life and Utopia*. Broadview Press Ltd, Peterborough, Ont, November 2005.
- [Sob15] Paweł Sobociński. Graphical Linear Algebra. <https://graphicallinearalgebra.net/>, June 2015.
- [Søg23] Anders Søgaard. Grounding the Vector Space of an Octopus: Word Meaning from Raw Text. *Minds & Machines*, 33(1):33–54, March 2023.
- [Spi] Spintronics - Build Mechanical Circuits. <https://upperstory.com/spintronics>.
- [Sut19] Richard Sutton. The Bitter Lesson, 2019.
- [Sza00] Zoltán Gendler Szabó. Compositionality As Supervenience. *Linguistics and Philosophy*, 23(5):475–505, October 2000.
- [ted22] teddy [@teddynpc]. I made ChatGPT take a full SAT test. Here's how it did: <https://t.co/734sPFU3HY>, December 2022.
- [TGZ⁺23] Taori, Rohan, Gulrajani, Ishaan, Zhang, Tianyi, Dubois, Yann, Li, Xuechen, Guestrin, Carlos, Liang, Percy, and Hashimoto, Tatsunori B. Stanford CRFM. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 2023.
- [Tho22] Alan D. Thompson. GPT-3.5 IQ testing using Raven’s Progressive Matrices, 2022.
- [Tom22] Tom Goldstein [@tomgoldsteincs]. Training PaLM takes 3.2 million kilowatt hours of power. If you powered TPUs by riding a bicycle, and you pedaled hard (nearly 400 watts), it would take you 1000 years to train PaLM, not including bathroom breaks. In that time, you'd make 320 trips around the globe!, July 2022.

- [Urq20] Alasdair Urquhart. Fine on Arbitrary Objects. In Mircea Dumitru, editor, *Metaphysics, Meaning, and Modality: Themes from Kit Fine*, page 0. Oxford University Press, October 2020.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017.
- [Wan19] Vincent Wang. Graphical Grammar + Graphical Completion of Monoidal Categories. 2019.
- [WC] Vincent Wang-Mascianica and Bob Coecke. Internal Wirings. *Springer LNCS*.
- [WC21] Vincent Wang-Mascianica and Bob Coecke. Talking Space: Inference from spatial linguistic meanings, September 2021. Comment: 33 pages, many pictures.
- [WGZ22] Paul Wilson, Dan Ghica, and Fabio Zanasi. String diagrams for non-strict monoidal categories, January 2022.
- [WHBW21] Matthew Wilson, James Hefford, Guillaume Boisseau, and Vincent Wang. The Safari of Update Structures: Visiting the Lens and Quantum Enclosures. *Electron. Proc. Theor. Comput. Sci.*, 333:1–18, February 2021. Comment: In Proceedings ACT 2020, arXiv:2101.07888.
- [Wie96] Anna Wierzbicka. *Semantics : Primes and Universals*. Oxford University Press UK, Oxford England ; New York, March 1996.
- [wik22] Jaquet-Droz automata. *Wikipedia*, November 2022.
- [WLC23] Vincent Wang-Mascianica, Jonathon Liu, and Bob Coecke. Distilling Text into Circuits, January 2023. Comment: 53 pages, lots of figures.
- [Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc, Champaign, IL, illustrated edition edition, August 2002.
- [WWS⁺23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, January 2023.
- [Yau16] Donald Yau. *Colored Operads*. American Mathematical Society, Providence, Rhode Island, February 2016.
- [YK21] Richie Yeung and Dimitri Kartsaklis. A CCG-Based Version of the DisCoCat Framework, May 2021. Comment: SemSpace 2021: Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science.

- [Zam17] Vladimir Nikolaev Zamdzhev. *Rewriting Context-free Families of String Diagrams*. PhD thesis, The University of Oxford, May 2017. Comment: PhD Thesis. Successfully defended in August 2016.
See PDF for full abstract.