VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

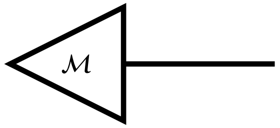
0

Synopsis

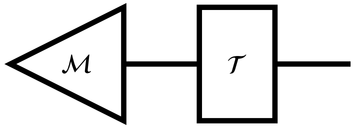
0.1 What is this thesis about?

THIS THESIS IS ABOUT A NEW WAY TO SEE LANGUAGE USING FORMAL DIAGRAMS

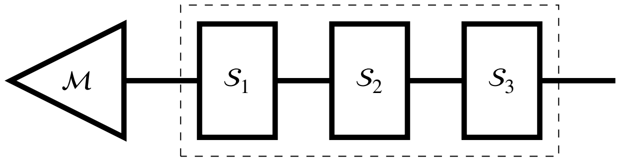
THE DIAGRAMS LOOK LIKE CIRCUITS.
Let's say that the meaning of text is how it updates a model.
So we start with some model of the way things are.



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

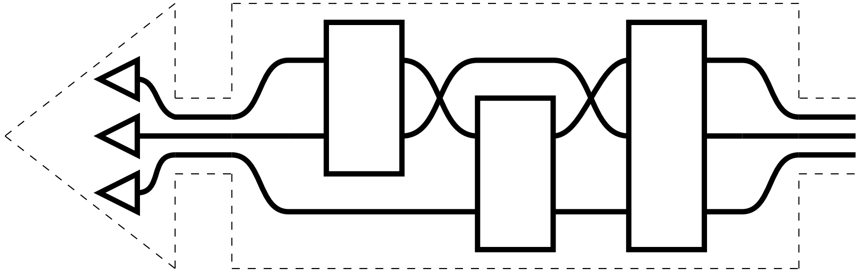


Text is made of sentences; like a circuit is made of gates and wires.

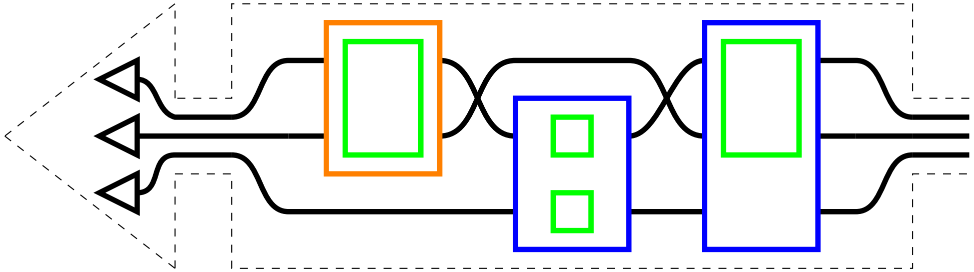


Let's say that ***The meaning of a sentence is how it updates the meanings of its parts.***
As a first approximation, let's say that the *parts* of a sentence are the nouns it contains or refers to.
Noun data is carried by wires.

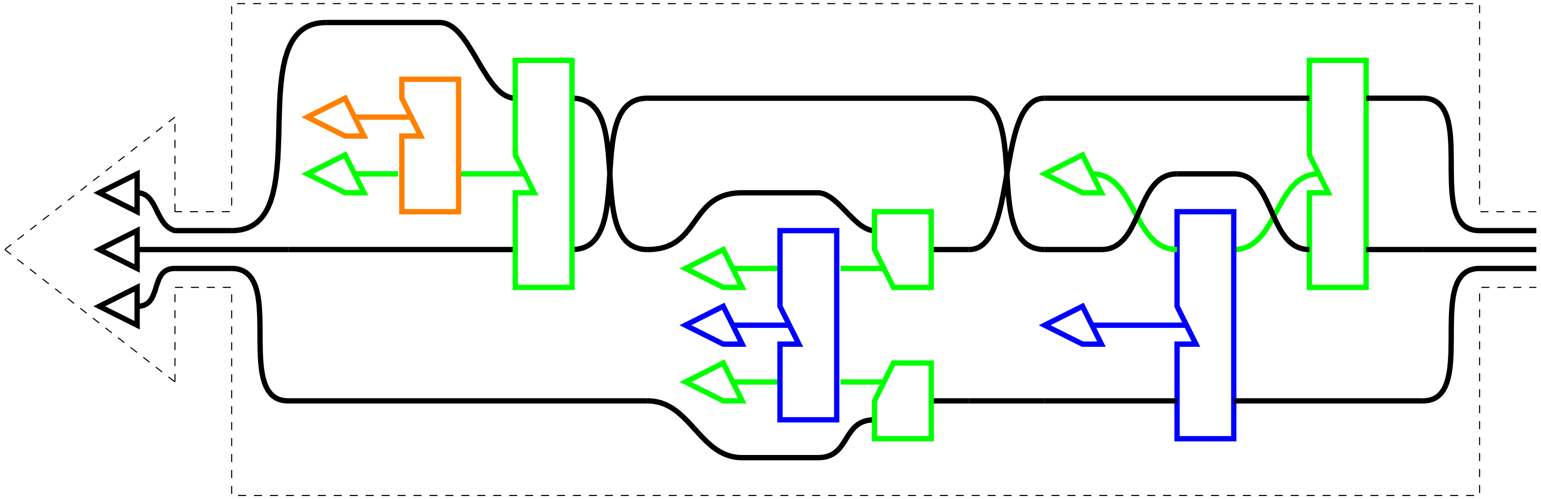
Collections of nouns are related by gates.



Gates can be related by higher order gates.



Higher order gates may be implemented as gates that modify the parameters of other gates.



Every gate corresponds to a *content word* – a word with a dictionary meaning.

placeholder

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

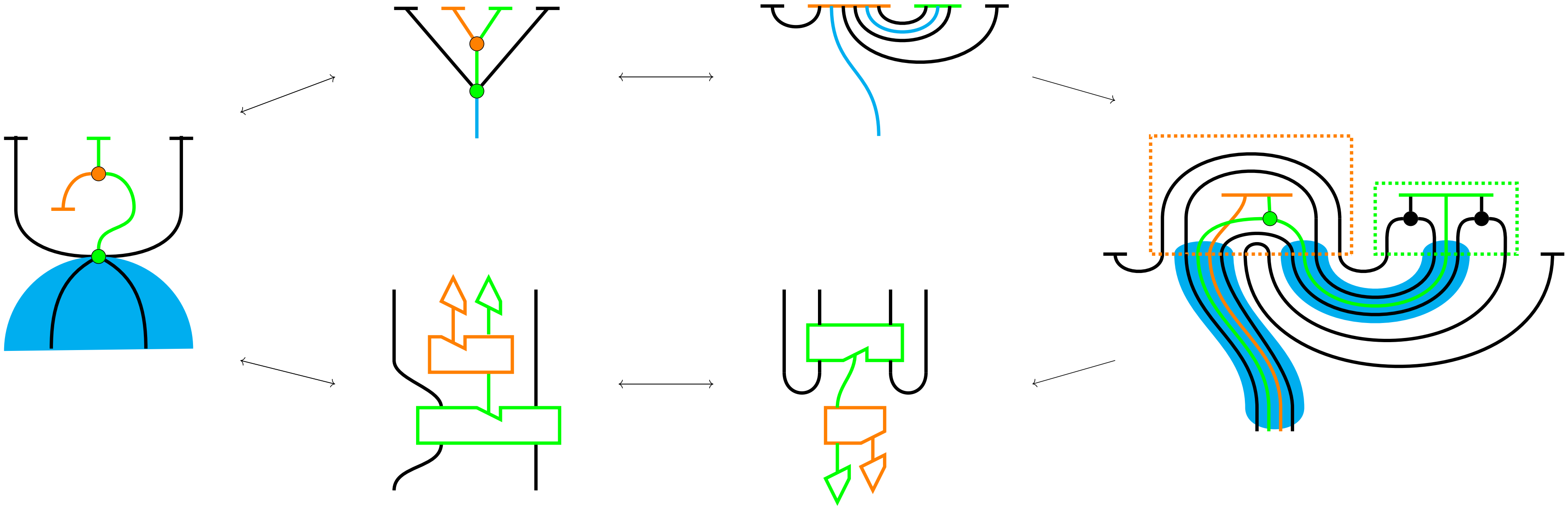
placeholder

TEXT DIAGRAMS ARE FORMAL COMPOSITIONAL BLUEPRINTS.
These compositional blueprints may be instantiated by classical or quantum computers.
We know how grammar composes meanings in language.
We can quotient out grammar using these formal diagrams.
It turns big black boxes into composites of small black boxes,
which leaves smaller pieces for machines to learn representations for.
We introduce the mathematics, history, and philosophy of these diagrams in Chapter 1.

0.2 What is the shape of language?

TEXT CIRCUITS MAKE DIFFERENT WAYS OF THINKING ABOUT LANGUAGE LOOK THE SAME
Category Theory is good at defining and bookkeeping structure.
Chapter 2 is about formal correspondences between different shapes of language.
We show how text circuits bridge several existing formalisms of language.

And we recount a proof of the expressive capacity of text circuits.



Discourse Representation Theory

Text is composed of sentences as circuits are composed from gates.
Text circuits are composed by connecting noun wires.
This can happen when two sentences refer to the same noun.
`Alice likes Bob.  Bob hates Claire.`

placeholder

Or when a pronoun is used to refer to another noun. `Alice likes the flowers that Bob gives Claire.`

placeholder

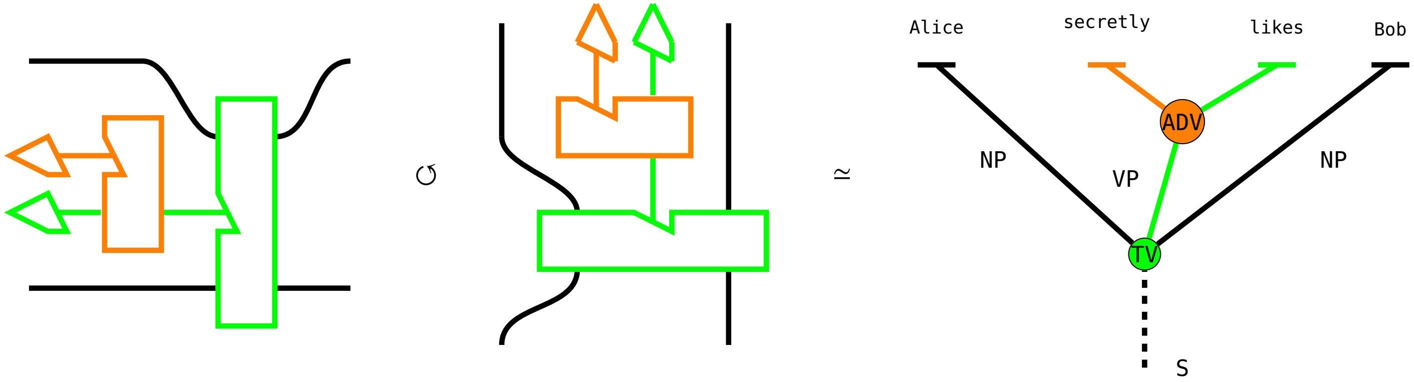
So if you know how to resolve pronoun references,
and you know how to represent sentences as gates acting on noun wires,
then you may represent text as circuits.
There are several ways to obtain gates from grammars for sentences.

Context-free and Tree-adjoining grammars

In a context-free grammar, a sentence symbol S is expanded by replacing individual symbols with strings of symbols to obtain a well-formed sentence.

The shape of these expansions is a tree.

That tree gives the shape of gates in a sentence.



In Chapter **??**, we characterise the generative grammar of text circuits in terms of a context-free grammar with additional structure.

Typelogical grammars

A typelogical grammar is like the inverse of a context-free grammar.

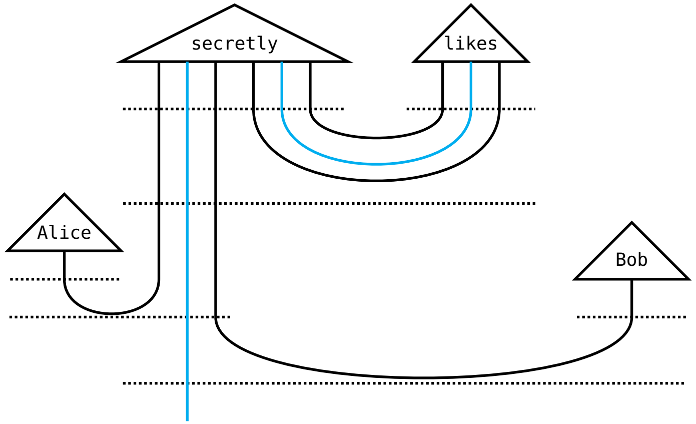
The latter is the grammar of the speaker: a full sentence is produced from S.

The former is the grammar of the listener: the sentence type s must be derived from the words of the sentence.

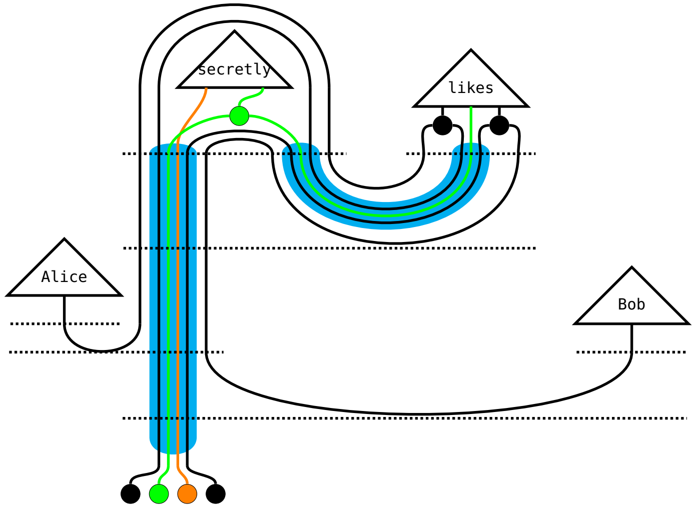
In a typelogical grammar such as a pregroup grammar, words are assigned types, and a logical proof witnesses the sentence type s:

$$\dfrac{\dfrac{\texttt{Alice} : n \qquad \dfrac{\texttt{secretly} : ({}^{-1}n\cdot s\cdot n^{-1})\cdot({}^{-1}n\cdot s\cdot n^{-1})^{-1} \qquad \texttt{likes} :{}^{-1} n\cdot s\cdot n^{-1}}{\texttt{secretly\textvisiblespace likes} :{}^{-1} n\cdot s\cdot n^{-1}}}{\texttt{Alice\textvisiblespace secretly\textvisiblespace likes} : s\cdot n^{-1}}\,[x^{-1}\cdot x\to 1] \qquad \texttt{Bob} : n}{\texttt{Alice\textvisiblespace secretly\textvisiblespace likes\textvisiblespace Bob} : s}\,[x\cdot^{-1} x\to 1]$$

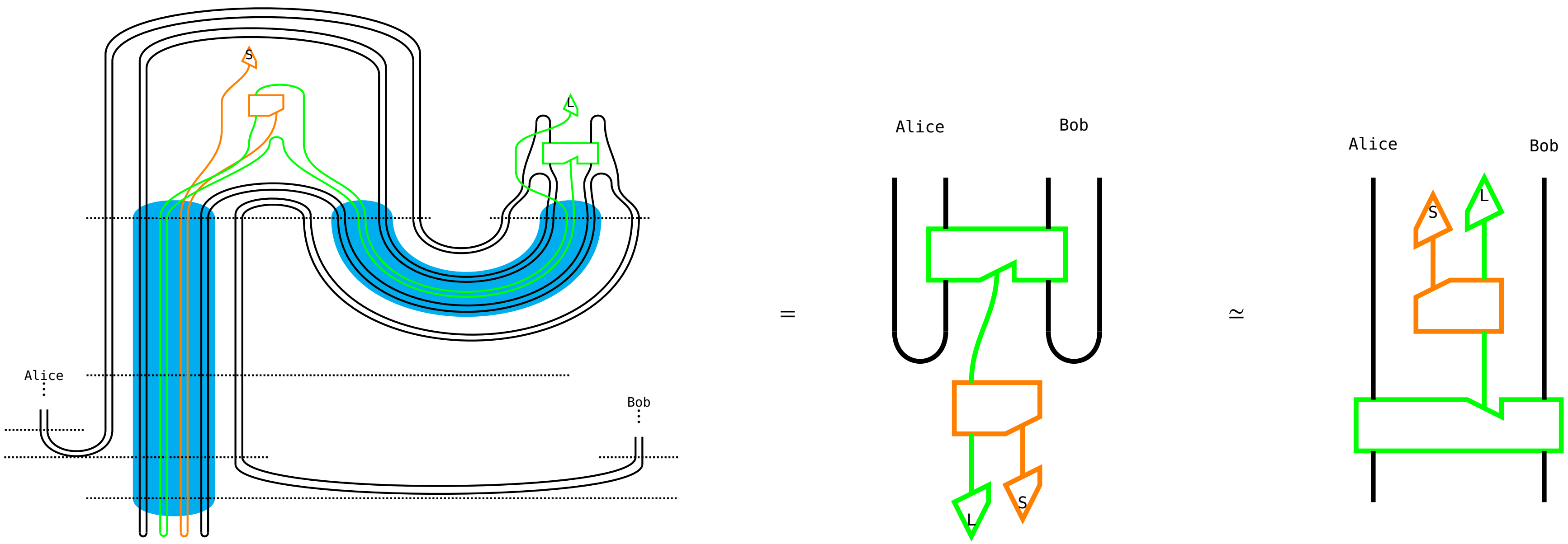
Such proofs of syntactic correctness can be re-expressed as diagrams:



To obtain a circuit, we make the sentence type s dependent on the words that occur in the sentence. To do this, wee refine the structure on each word by introducing *internal wirings* for word-states. Internal wirings make use of multiwires called spiders.



A certain choice of spider make these internal wiring diagrams topologically equivalent to the circuits we want.



Texts with the same meaning become the same circuit

0.3 *What new things can we do?*

In Chapter

0.4 *Other questions, such as:*

0.4.1 *Why is this thesis landscape?*

Why should it be otherwise? Screens are landscape. My diagrams are needy of space, unlike text. Fat margins for formal and informal asides.

0.4.2 What is this thesis not about?

This thesis is about a particular idealised and computer-friendly conception of natural language syntax and semantics, stated using the as-of-yet uncommon mathematical dialect of applied category theory, and informed by natural language but bound in intent to empirical capture. This means there is no serious consideration of phonetics, phonology, morphology, pragmatics, and the historical, social and psychological dimensions of language. Text circuits do not provide grammaticality judgements upon one-dimensional strings of language, so in some sense they are not even a theory of syntax. There is, moreover, no correct way to instantiate the abstract gates of a text circuit, so in some sense they are not even a theory of semantics.[1]

While the application of this mathematical perspective to the puzzles of natural language is relatively fresh, there are no new mathematical techniques developed here. Nominally the most complicated mathematical gadgetry used will be finitely presented associative n-categories, but they are mostly useful to us as a combinatorial handle on symmetric monoidal categories with regions and rewrites, in the same way one would use a PROP with relations for just symmetric monoidal categories.

There will be no code, no demonstrations, nothing practical. I am only concerned with showing that certain things are achievable in principle here, and how they may be achieved. Besides, since this is fundamental research, I can think of no demonstration that can outshine the state-of-the-art.

0.4.3 Why should I read this?

I wouldn’t want to be a formal linguist today. What would linguistics look like if it began today? Large language models (LLMs) such as GPT and PaLM would appear to us as oracles; wise, superhumanly capable at language, but inscrutable. Similar to how most people effortlessly use language without a formal understanding which they can express. So the fundamental mystery would remain: we would still seek to understand language, because understanding LLMs is completely different from understanding language[2].

So the presence of LLMs should not totally discourage our endeavour to understand language, but their sheer ability to do impressive things presents strong constraints about the acceptability of theories of language, by the usual standards of veridicality and utility[3]. In constast to the idealised and partial nature of formal theories, the nature of LLMs is that they are trained on empirical data about language that captures the friction of the real world. So, in terms of raw predictive power, we should naturally expect the LLMs to have an advantage over principled theories[4]. Here is a problem: if the better theory is one that lets you do more things, many theories of language are knocked out of the game by an LLM before they can even get started.

There are other, innate, tensions to playing the game of formal linguistics. I struggle to reconcile the empirical and scientific character to the study of language with the lack of usefulness; evidence of the success of the physical

[1] So how are text circuits about syntax and semantics? Text circuits are a "natural" metasyntax for natural language. We start from two assumptions. First, that linguistic meanings are compositional – after all, we have finite dictionaries for words but not for sentences, and we can understand infinitely many novel sentences – and second, that the syntax of a sentence (whatever that might be) directs the composition of the meaning of the words. In Section 1.4.2, we see that taking syntax to be operadic (tree-shaped) and composition to be via lambda calculus, we arrive at Montague semantics. Text circuits arise by generalising from operadic syntax to more expressive shapes (circuit-shaped), and specialising from Turing-complete lambda calculus to interpretation in symmetric monoidal categories, a constrained family of settings better suited to representing and reasoning about interacting computational or physical processes. The consequences, elaborated in the same section, are that superficial differences in the surface form of text are eliminated: text that mean the same, look and compute the same.

[2] Suppose you knew the insides of a mechanical calculator by heart. Does that mean you *understand* arithmetic? At best, obliquely. Implementing ideal arithmetic means compromises; the calculator is full of inessentialities and tricks against the constraints of physics. You would not know where the tricks begin and the essence ends. Suppose you knew every line of code and every piece of data used to train an LLM. How could one delineate what is essential to language, and what is accidental?

[3] The explanatory value of a theory by itself – the "aha!" – is comparable to the aesthetic pleasure of art, but we can make up such stories all day. Explanatory value paired with veridicality and utility is the gold standard.

[4] Just as the Ptolemaic geocentric model of the solar system was more empirically precise than the heliocentric Copernican, though the latter was "more correct". This was because Ptolemaic epicycles can overfit to approximate any observed trajectory of planets. It took until Einstein’s relativity to explain the perihelion of mercury, which at last aligned theoretical understanding with empirical observation.

sciences are all around us, but where are the practical applications of formal linguistics? I have certainly felt like I was doing a kind of empirical observation by consulting my own acceptability judgements and intuitions when constructing models of English grammar. However, the laws and structure of language are smothered by a nebulous foliage of "unless" and "it depends" and other hedges. Probing for the elusive empirical truth of language by the objective process of asking others for grammaticality judgements helps a little, until you start to agree all the time, at which point there is a suspicion we could be fooling ourselves. To summarise: nobody can be sure whether they are truly isolating the essence of language or bullshitting, and to top it off, nowadays either case is irrelevant in practice!

I hope to have convinced you of the need for meaningful standards to pursue formal linguistics by, in these depressing circumstances. I have been tempted to give up and treat the activity of formal linguistics as a form of art; harmless doodling for its own sake. But theorybuilding is distinguishable from storytelling by other standards beyond veridicality and utility. I have tried to hold myself to the following, which I think are reasonable:

- **If you can't beat them, make room for them.** There is value in synthesis. If your theory of language can neither empirically outperform nor work in tandem with neural methods in principle, it is a practical nonstarter.

- **If you can't beat them, be simpler.** There is value in a theory that provides unified understanding even if it is simplified, practically yet unrealised, or momentarily empirically embarrassed.

- **If you can't beat them, play a different game.** There is value in breaking new ground and extending our reach. If your theory of language can't make room and isn't simple and also has no force of originality beyond the reach of an LLM, it is boring.

- **If you can't beat them, smile and look pretty.** There is value in art. But if your theory of language is practically inferior, complicated, does nothing new, and on top of that it's *ugly*? You don't have a linguistic theory, you have a conspiracy theory.

IF I HAD TO BE A FORMAL LINGUIST TODAY, I WOULD USE APPLIED CATEGORY THEORY. Our capacity for language is one of the oldest and sophisticated pieces of compositional technology; maybe even the foundation of compositional thought. So, linguists are veteran students of compositionality and modularity. How does syntax compose meaning? How do the constraints and affordances of language interact? The discipline embodies a encyclopaedic record of how compositionality works "in the field", just as botanists record flowers, early astronomers the planetary motions, or stamp collectors stamps.

Here I argue that most formal linguists have a metalanguage problem. Set-theoretical foundations of mathematics are not well suited for complex and interacting moving parts. Let's call such systems *jungle-systems*. To specify a set-theoretic model necessitates providing complete detail of how every part looks on the inside[5] – because only

[5] This is a foundational, innate problem of set theory. Consider the case of the cartesian product of sets, one of the basic constructions. $A \times B$ is the "set of ordered pairs" (a, b) of elements from the respective sets, but there are many ways of encoding that are equivalent in spirit but not in syntax; a sign that the syntax is a hindrance. What we really want of the

then can you define how the functions map elements to elements. Abstraction becomes unnecessarily difficult, which becomes a hindrance when working with jungle-systems. Computer scientists study abstractions, and being one, I would say that having good abstractions means easier debugging and interoperability – the entirety of Chapter 2 is an example that shows how coding different systems in the right metalanguage makes it easy to compare, relate, and extend those theories. This jungle-native metalanguage is (Applied) Category Theory. This train of thought is continued in Section 1.2.4.

0.4.4 What's wrong with our historical tools?

Here is a fair challenge from a hypothetical formal semanticist: *"We already have a general-purpose metalanguage solution for composition: the lambda-calculus. What can we do with category theory that we cannot already do?"* Shortest answer: Punchcard machines are turing-complete, so the answer is that there is nothing that a high-level language like python on a modern laptop can compute that a punchcard machine cannot, and we are all free to program how we like. A more constructive and historically-situated answer is in Section 1.4.

0.4.5 What are the novel contributions?

- A brief characterisation of Montague's conception of syntax in modern mathematical terminology, and diagrammatically.
- A theory of text structure compatible with quantum and classical data-driven modelling.
- A structural correspondence between tree-adjoining grammars, typelogical grammars, and discourse representation theories.
- A recapitulation of the characterisation of text circuits by a controlled fragment of English from [longpaper], this time including the formal n-categorical signatures.
- A string-diagrammatic presentation of a category of continuous relations, which serves to enable concrete calculations for the following:
  - Formal semantics for interacting spatially-embodied agents.
  - A method to formally model textual metaphors.
  - A linguistic characterisation of o-minimal structures.

1

Background

1.1 A Partial History of String Diagrams

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

String diagrams are a loophole in the devil's contract. Descartes cast geometry as algebra, and string diagrams do the reverse.

WE HAVE ALWAYS TRIED TO CAPTURE LANGUAGE IN DIAGRAMS.

Every so often, someone tries once again to draw the shape of language.

...

Now it's our turn.

The difference this time is formality; we are using *string diagrams*.

STRING DIAGRAMS ARE FORMAL REASONING AND REPRESENTATION SYSTEMS FOR MONOIDAL CATEGORIES. Chronologically, those adjectives occurred in this order: 1) visual representation, 2) visual reasoning, 3) formal.

1.1.1 Pre-formal Diagrams

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

Euclid: Geometry

Venn & Carroll: Syllogisms

Petri: Chemistry

1.1.2 Convergent Evolution

String diagrams arise naturally as diagrammatic representations for formal theories where reoccurring variables or bookkeeping indices for keep track of pairwise connections become too unwieldy for one-dimensional syntax.

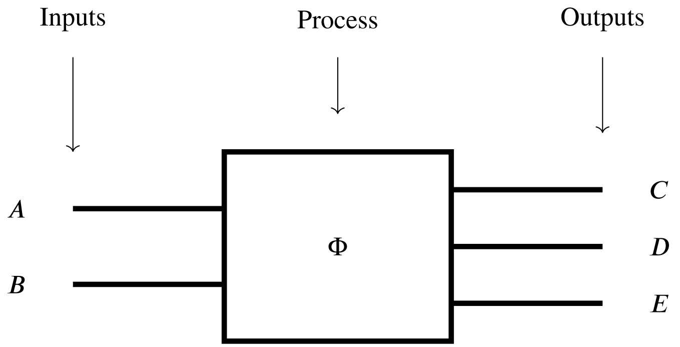
IN PHYSICS

In Logic

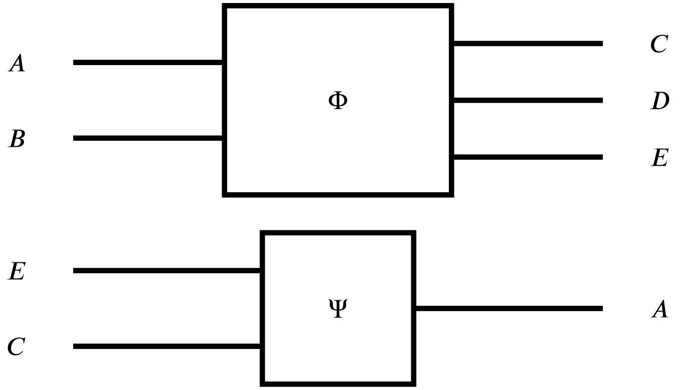
In Computer Science

1.2 Process Theories

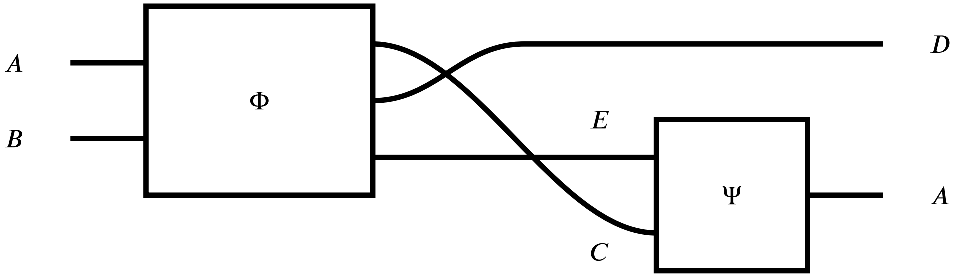
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. We read processes from left to right.



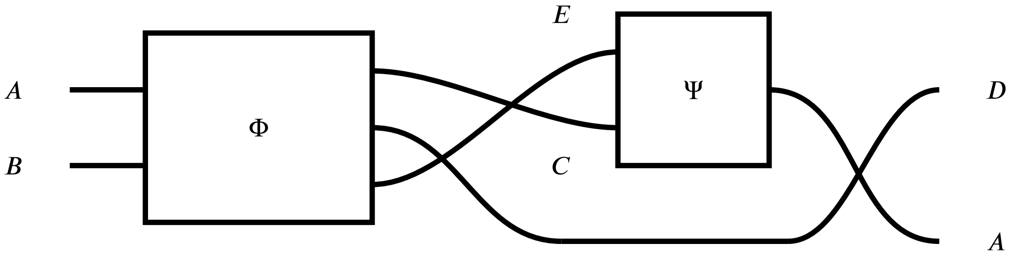
Processes may compose in parallel, which we depict as vertically stacking boxes.



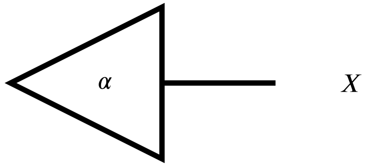
Processes may compose sequentially, which we depict as connecting wires of the same type from left to right.



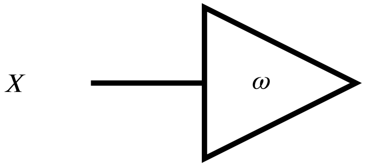
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



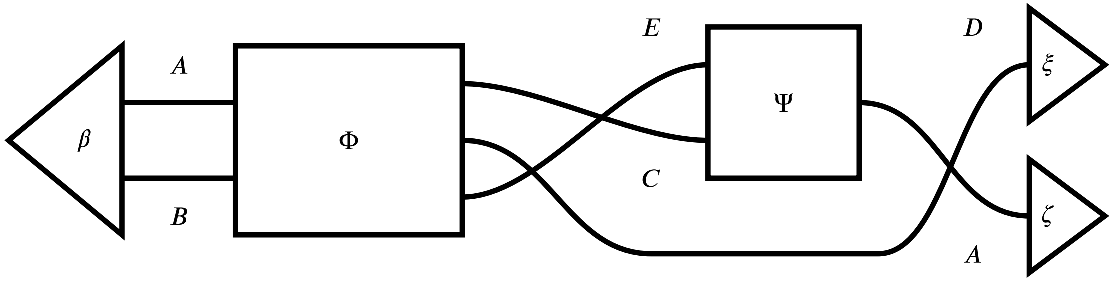
Some processes have no inputs; we call these *states*.



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



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



A process theory is given by the following data[1]:

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

Example 1.2.1 (Linear maps with direct sum)**.** Systems are finite-dimensional vector spaces over R. Processes are linear maps, expressed as matrices with entries in R.

Sequential composition is matrix multiplication. Parallel composition of systems is the direct sum of vector spaces $\oplus$. The parallel composition of matrices $\mathbf{A}, \mathbf{B}$ is the block-diagonal matrix

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

The unit of parallel composition is the singleton 0-dimensional vector space. States are row vectors. Tests are column vectors. The numbers are R.[2]

Example 1.2.2 (Sets and functions with cartesian product)**.** Systems are sets A, B. Processes are functions between sets $f : A \to B$. Sequential composition is function composition. Parallel composition of systems is the cartesian product of sets: the set of ordered pairs of two sets.

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

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

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

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

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

The unit of parallel composition is the[3] singleton set $\{\star\}$. States of a set A correspond to elements $a \in A$[4]. Every system A has only one test $a \mapsto \star$[5]. There is only one number.

Example 1.2.3 (Sets and relations with cartesian product)**.** Systems are sets A, B. Processes are relations between sets $\Phi \subseteq A \times B$, which we may write in either direction $\Phi^* : A \nrightarrow B$ or $\Phi_* : B \nrightarrow A$.[6] Sequential composition is relation composition:

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

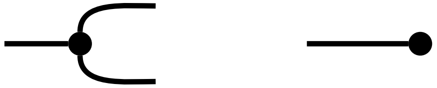
Parallel composition of systems is the cartesian product of sets. The parallel composition of relations $A \otimes C \overset{\Phi \otimes \Psi}{\nrightarrow} B \otimes D$ of relations $A \overset{\Phi}{\nrightarrow} B$ and $C \overset{\Psi}{\nrightarrow} D$ is defined:

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

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

1.2.1 *What does it mean to copy and delete?*

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



Example 1.2.4 (Linear maps)**.** Consider a vector space $\mathbf{V}$, which we assume includes a choice of basis. The copy map is the rectangular matrix made of two identity matrices:

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

The delete map is the column vector of 1s:

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

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

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

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

[6] Relations between sets are equivalently matrices with entries from the boolean semiring. Relation composition is matrix multiplication with the boolean semiring. Φ^*, Φ_* are the transposes of one another.

Example 1.2.5 (Sets and functions)**.** Consider a set A. The copy function is defined:

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

The delete funtion is defined:

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

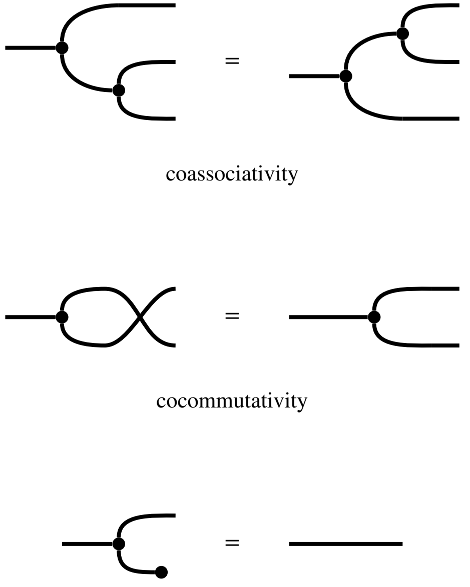
Example 1.2.6 (Sets and relations)**.** Consider a set A. The copy relation is defined:

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

The delete relation is defined:

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

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



coassociativity

cocommutativity

counitality (1.1)

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

Coassociativity: says there is no difference between copying copies.

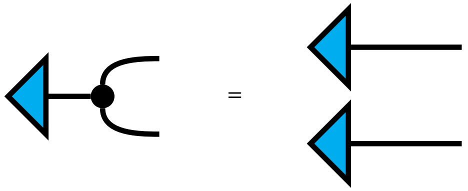
Cocommutativity: says there is no difference between the outputs of a copy process.

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

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

Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 1.2.7 and Remark 1.2.8, we know that this demand is incompatible with certain process theories. In particular, this demand would constrain a process theory of sets and relations to a subtheory of sets and functions. The moral here is that process theories are flexible enough to meet ontological needs. A classical computer scientist who works with perfectly copyable data and processes might demand universal copying along with Equations 1.1, whereas a quantum physicist who wishes to distinguish between copyable classical data and non-copyable quantum data might taxonomise copy and delete as a special case of a more generic quasi-copy and quasi-delete that only satisfies equations 1.1.[7]

Example 1.2.7 (Not all states are copyable)**.** Call a state *copyable* when it satisfies the following diagrammatic equation:



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

1.2.2 What is an update?

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

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

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

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

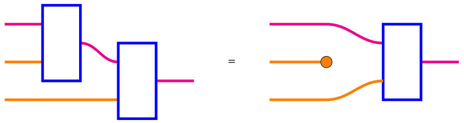
That was a flash-prehistory of *bidirectional transformations* []. Following the monoidal generalisation of lenses in [], a rewrite as we have described above is specified by system diagrammatic equations in the margin, each of which we introduce in prose.

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

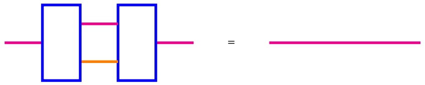
A *kind of process* is determined by patterns of interaction with other kinds of processes.

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

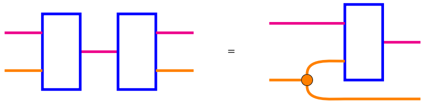
PutPut: Putting in one value and then a second is the same as deleting the first value and just putting in the second.



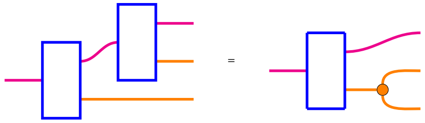
GetPut: Getting a value from a field and putting it back in is the same as not doing anything.

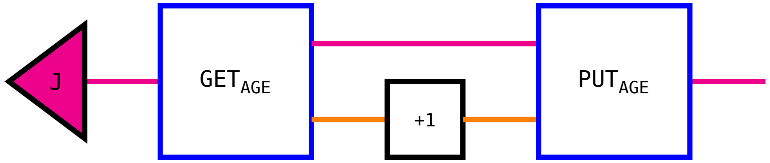


PutGet: Putting in a value and getting a value from the field is the same as first copying the value, putting in one copy and keeping the second.



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





1.2.3 Spatial predicates

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

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

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

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

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

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

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

$$placeholder$$

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

$$placeholder$$

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

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

$$placeholder$$

1.2.4 Some basic properties of linguistic spatial relations

1.3 Defining String Diagrams

1.3.1 Symmetric Monoidal Categories

1.3.2 PROPs

1.3.3 1-object 4-categories

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

1.4 A brief history of formal linguistics from the categorial perspective

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

1.4.1 Curry-Howard-Lambek

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

Describing the bigger benefit requires a bigger picture.

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

Or this:

So here is the story today as far as a linguist may be concerned. We know that Curry-Howard-Lambek- correspondence generalises: if you poke Howard for a grammar that is expressively distinct from a CCG, the type-theory

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

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

changes, so Lambek gives a different family of semantic categories with different internal logics, and Curry gives you a syntactic composition gadget that differs from the lambda-calculus.

1.4.2 What did Montague consider grammar to be?

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

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

There is a natural division of Montague's approach into two structural components. First, the notion of a structure-preserving map from syntax to semantics. Second, the use of a powerful and expressive logic for semantics. We acknowledge the importance of the latter for formal semantic engineering, but here we will focus on just the former. According to Partee [], a formal semanticist, advocate, and torch-bearer for Montague, the chief interest of Montague's approach (as far as his contemporary *linguists* were concerned) lay in the following ideas:

1. Take truth conditions to be the essential data of semantics.
2. (a) Use lambdas to emulate the structure of syntax...

[8]

[9]

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

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

In Section 1 of *Universal Grammar*, Montague’s first paragraph establishes common notions of relation and function – the latter he calls *operation*, to distinguish the *n*-ary case from the unary case which he calls *function*. This is all done with ordinals indexing lists of elements of an arbitrary but fixed set A, which leads later on to nested indices and redundancy by repeated mention of A. We will try to avoid these issues going forward by eliding some data where there is no confusion, following common modern practice.

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

Definition 1.4.2 is equivalent to asking for all projections. Definitions 1.4.2 and 1.4.4 together characterise Montagovian algebras as (concrete) clones [].

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

1.4.3 On Syntax

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

1. $(\delta \ \in)\Delta$ is an (indexing) set of syntactic categories (e.g. `NP`, `V`, etc.). Montague calls this the *set of category indices*. $X_\delta \subseteq A$ form the *basic expressions* of type δ in the language.
2. a set S assigns types among $\delta \in \Delta$ to the inputs and output of – not necessarily all – F_γ.
3. a special $\delta_0 \in \Delta$ is taken to be the type of declarative sentences.

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

Taking a disambiguated language $\mathfrak{U}$ on a carrier set A, Montague defines a *language* to be a pair $L :=< \mathfrak{U}, R >$, where R is a relation from a subset of the carrier A to a set $\mathtt{PE}_L$, the set of *proper expressions* of the language L. An

Definition 1.4.1 (Generating data of an Algebra)**.** Let A be the carrier set, and F_γ be a set of functions $A^k \to A$ for some $k \in N$, indexed by $\gamma \in \Gamma$. Denoted $\langle A, F_\gamma \rangle_{\gamma\in\Gamma}$

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

Definition 1.4.2 (Identities)**.** A family of operations populated, for all $n, m \in N$, $n \leq m$, 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 1.4.3 (Constants)**.** For all elements of the carrier $x \in A$, and all $m \in N$, a constant operation $C_{x,m}$ defined on all m-tuples a as:

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

Definition 1.4.4 (Composition)**.** Given an n-ary operation G, and n instances of m-ary operations $H_{1\leq i\leq n}$, define the composite $G(H_i)_{1\leq i\leq n}$ to act on m-tuples a by:

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

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

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

Definition 1.4.5 (Polynomial Operations)**.** The polynomial operations over an algebra $\langle A, F_\gamma \rangle_{\gamma\in\Gamma}$ are defined to be smallest class K containing all $F_{\gamma\in\Gamma}$, identities, constants, closed under composition.

Definition 1.4.6 (Homomorphism of Algebras)**.** h is a homomorphism from $\langle A, F_\gamma \rangle_{\gamma\in\Gamma}$ *into* $\langle B, G_\gamma \rangle_{\gamma\in\Delta}$ iff

1. $\Gamma = \Delta$ and $\forall \gamma$:
2.

admirable purpose of R appears to be to permit the modelling of *syntactic ambiguity*, where multiple elements of the term algebra $\mathfrak{U}$ (corresponding to syntactic derivations) are related to the same 'proper language expression'.

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

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

2

String Diagrams for Text

2.1 How do we communicate?

SPEAKERS AND LISTENERS UNDERSTAND ONE ANOTHER. As far as formal theories of syntax and semantics go, this is a miracle. It remains so even if we cautiously hedge this mundane statement to exclude pragmatics and context and only encompass small and boring fragments of factual language.

In short: there is a distinction between *grammars of the speaker* – which produce sentences – and *grammars of the listener* – which deduce sentences. Viewed as mathematical processes, the two kinds of grammars go in opposite directions; speaking grammars [e.g. string-rewrite systems] start with some structure, and require informational input [e.g. which rule comes next] to produce sentences; listening grammars [e.g. typelogical grammars] start with a sentence, and require informational input [e.g. grammatical typing and which proof rule to try next] to deduce a grammatical structure. Since we can understand each other, these two types of grammar must enjoy a systematic correspondence. The correspondence looks something like this:

$$placeholder$$

In this section I will elaborate on the above argument, and provide the first steps of a formal mathematical framework to present and organise the nature of this correspondence. This framework comes from the idea that speakers and listeners may use language as a vehicle to communicate thought; from this formal framework, we detect that the intermediate structure – that which is being communicated – leads us to the idea of text circuits.

The empirical observation that speakers and listeners can communicate is a trivial one; any account of language that does not take this interactive and procedural aspect into account is arguably lacking. Text circuit theory aims to provide the beginnings of an theory of language that accounts for not just this...

2.1.1 Speaking grammars, listening grammars

Here are some naïve observations on the nature of speaking and listening. Let's suppose that a speaker, Preube, wants to communicate a thought to Fondo. Just as a running example that does not affect the point, let's say we can gloss the thought as carrying the relations:

(alice, bob, flowers, like, give)

Preube and Fondo cooperate to achieve a minor miracle; Preube encodes his thoughts – a structure that doesn't look like a one-dimensional string of symbols – into a one-dimensional string of symbols, and Fondo does the reverse, turning a one-dimensional string of symbols into *a thought-structure like that of Preube's.* The two can do this for infinitely many thoughts, and new thoughts neither have encountered before.

What I mean by this is just that both Preube and Fondo agree on the structure of entities and relations up to the words for those entities and relations. For example, Preube could ask Fondo comprehension questions such as `WHO GAVE WHAT? TO WHO?`, and if Fondo can always correctly answer – e.g. `BOB GAVE FLOWERS. TO CLAIRE.` – then both Preube and Fondo agree on the relational structure of the communicated thought to the extent permitted by language. It may still be that Preube *placeholder*

We assume Preube and Fondo speak the same language, so both know how words in their language correspond to putative building blocks of thoughts, and how the order of words in sentences and special grammatical words affect the (de/re)construction procedures.

placeholder

The nature of the problem can be summarised as an asymmetry of information. The speaker knows the structure of a thought and has to make choices to turn that thought into text. The listener knows only the text, and must make choices to deduce the thought behind it. By this perspective, language is a shared and cooperative strategy to solve this (de/en)coding interaction. I will now outline the constraints imposed by this interaction, and then explain how context-free grammars and typelogical grammars partially model these constraints.

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

`ALICE LIKES THE FLOWERS BOB GIVES CLAIRE.`

`BOB GIVES CLAIRE FLOWERS. ALICE LIKES THE FLOWERS.`

`THE FLOWERS TO CLAIRE THAT BOB GIVES ARE LIKED BY ALICE.`

Whether those decisions are made by committee or coinflips, those decisions represent information that must be supplied to Preube in the process of speaking a thought.

placeholder

For this reason, we consider context-free-grammars (and more generally, other string-rewrite systems) to be *Grammars of the speaker*. The start symbol S is incrementally expanded and determined by rule-applications that are selected by the speaker. The important aspect here is that the speaker has an initial state of information S that requires more information as input in order to arrive at the final sentence.

Listeners deduce. The listener Fondo must supply decisions about which words are grammatically related, and how. Like right-of-way in driving, sometimes these decisions are settled by convention, for example, subject-verb-object order in English. Sometimes sophisticated decisions need to be made that override or are orthogonal to conventions, and we give two examples.

Example 2.1.2 (Garden path sentences)**.** So-called "garden path" sentences illustrate that listeners have to make choices to resolve lexical ambiguities. One such garden-path sentence is `The old man the boat`, where typically readers take `The old man` as a noun-phrase and `the boat` as another noun-phrase. We can sketch how the readers might think with a (failed) pregroup grammar derivation:

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

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

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

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

Example 2.1.3 (Ambiguous scoping)**.** Consider the following sentence:

`Everyone loves someone`

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

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

which is stronger than the former. The last sentence is awkward: unlike in the first two cases, the quantified variable in the (gloss) $\cdots \neg\exists x_{Person} \cdots$ of `God knows` is lost, and what is left is a literal reading $\cdots \neg$`knows(God,` $\cdots$`)` $\cdots$. Second, whether a sentence is grammatically acceptable may depend on whether an exclamation is involved. `God knows` and `who knows` can be shuffled into the sentence to behave as an intensifier as in:

`Bob drinks God knows how many beers`

`Bob drinks who knows how many beers`

But it is awkward to have:

`Bob drank nobody knows how many beers`

And it is not acceptable to have:

`Bob drank Alice knows how many beers`

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

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

$$\llbracket\texttt{everyone}\rrbracket = \lambda(\lambda x.V(x)).\forall x : V(x) \tag{2.1}$$

$$\llbracket\texttt{loves}\rrbracket = \lambda x \lambda y.\texttt{loves}(x, y) \tag{2.2}$$

$$\llbracket\texttt{someone}\rrbracket = \lambda(\lambda y.V(y)).\exists y : V(y) \tag{2.3}$$

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

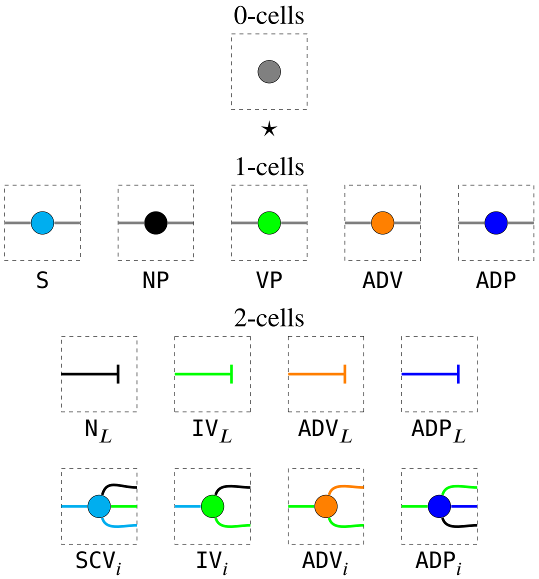
$$\dfrac{\dfrac{\lambda(\lambda x.V(x)).\ulcorner\forall x : V(x)\urcorner : (n \multimap s) \multimap s \qquad \lambda x\lambda y.\ulcorner\texttt{loves}(x, y)\urcorner : n \multimap s \multimapinv n}{\lambda y.\ulcorner\forall x : \texttt{loves}(x, y)\urcorner : s \multimapinv n} \qquad \lambda(\lambda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

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

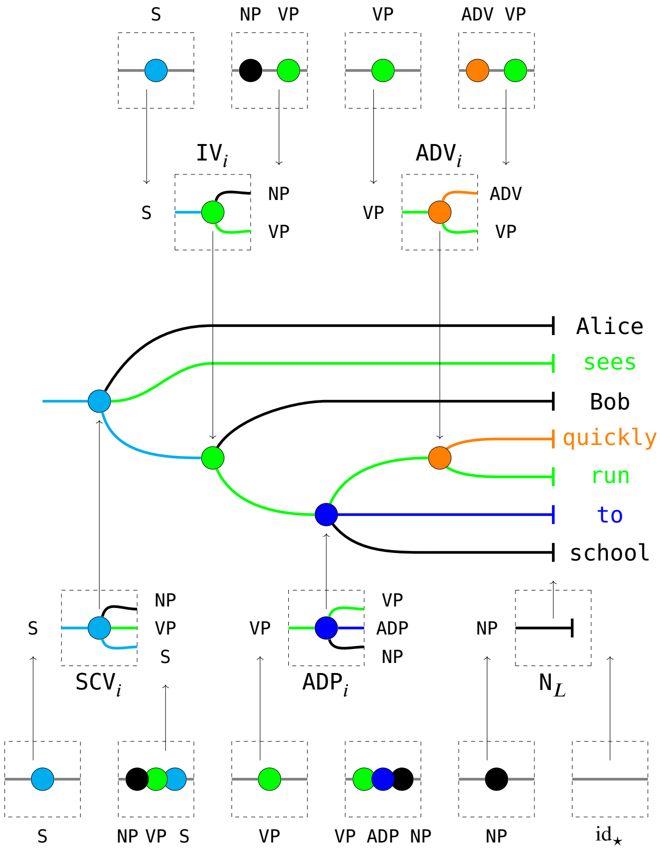
THE SPEAKER'S CHOICES AND THE LISTENER'S DEDUCTIONS MUST BE RELATED. Even if the pair are cooperating, so that the speaker tries to avoid ambiguity,

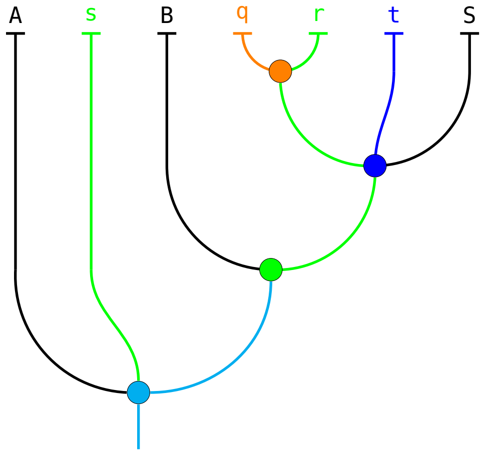
2.1.2 *A context free grammar to generate* `Alice sees Bob quickly run to school`

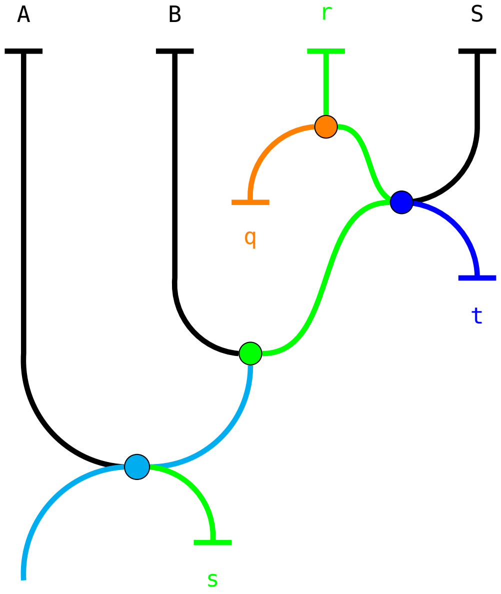
We can describe a context-free grammar with the same combinatorial rewriting data that specifies string diagrams. For this example, we take the following n-categorical signature. The generators subscripted L (for *label* or *leaf*) correspond to terminals of the CFG, and represent a family of generators indexed by a lexicon for the language. The generators subscripted i (for *introducing a type*) correspond to rewrites of the CFG.



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

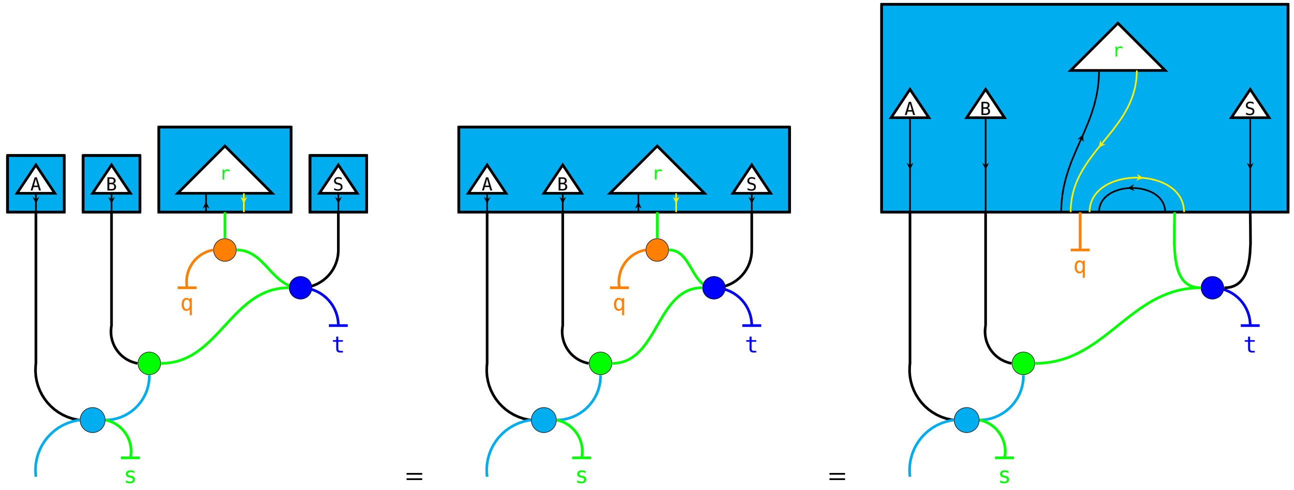

S
NP VP
VP
ADV VP
IV$_i$
ADV$_i$
S
NP
VP
VP
ADV
VP
Alice
sees
Bob
quickly
run
to
school
NP
S
VP
S
SCV$_i$
VP
VP
ADP
NP
ADP$_i$
NP
N$_L$
S
NP VP S
VP
VP ADP NP
NP
id$_\star$


A
s
B
q
r
t
S

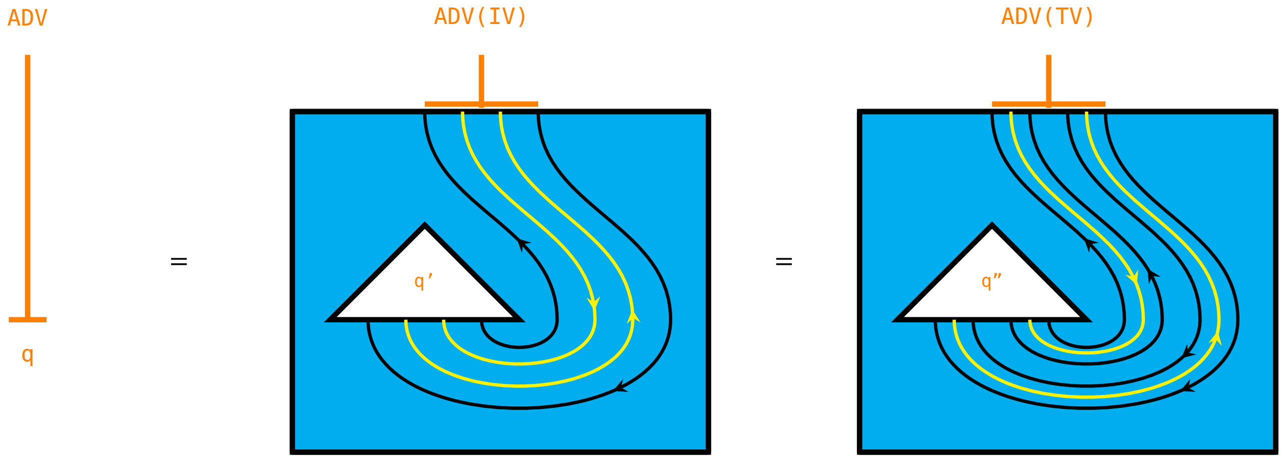

A
B
r
S
q
t
s

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

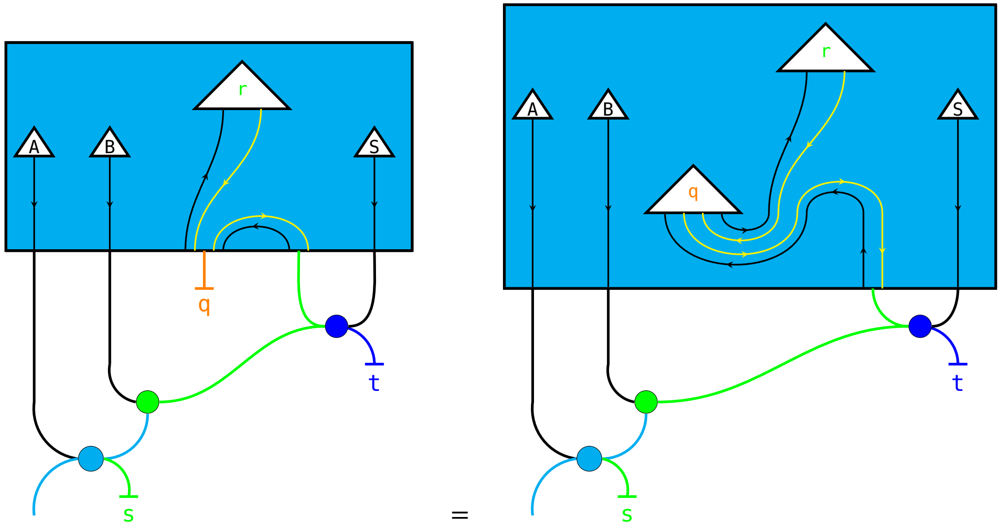
We merge the monoidal functor-boxes and we slide the bottom edge down using the fibration.



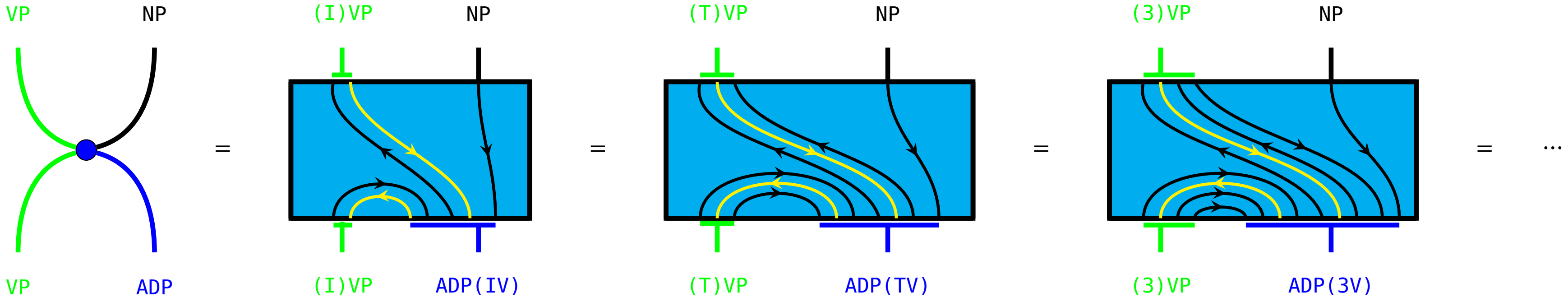
`quickly` could find itself modifying an intransitive (single noun) or transitive (two noun) verb. Suppose that it is the job of some process q′ to handle intransitive verbs, and similarly q″ to handle transitive ones. We use the functor for bookkeeping, by asking it to send both q′ and q″ to the dependent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



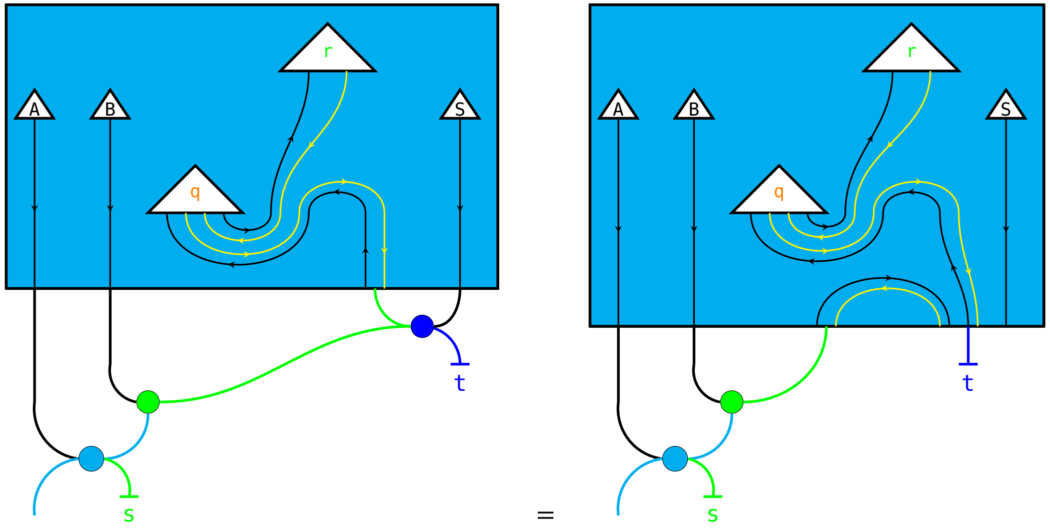
Since the functor is a monoidal discrete fibration, it introduces the appropriate choice of `quickly` when we pull the functor-box down, while leaving everything else in parallel alone.



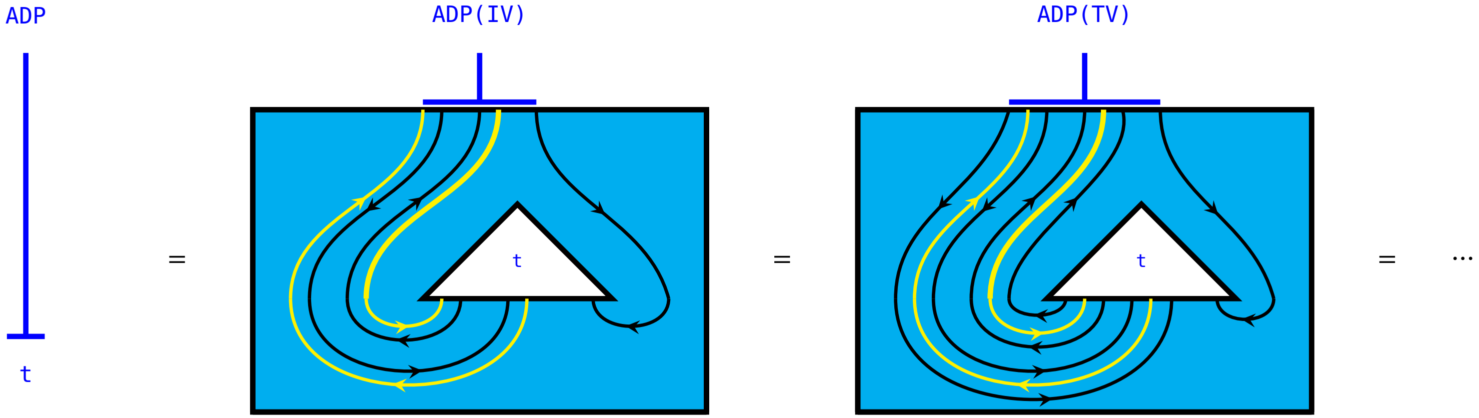
Adpositions can apply for verbs of any noun-arity. We again use the fiber of the functor for bookkeeping by asking it to send all of the following partial pregroup diagrams to the adposition generator. We consider the pregroup typing of a verb of noun-arity $k \geq 1$ to be ${}^{-1}n \cdot s \cdot \underbrace{n^{-1}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



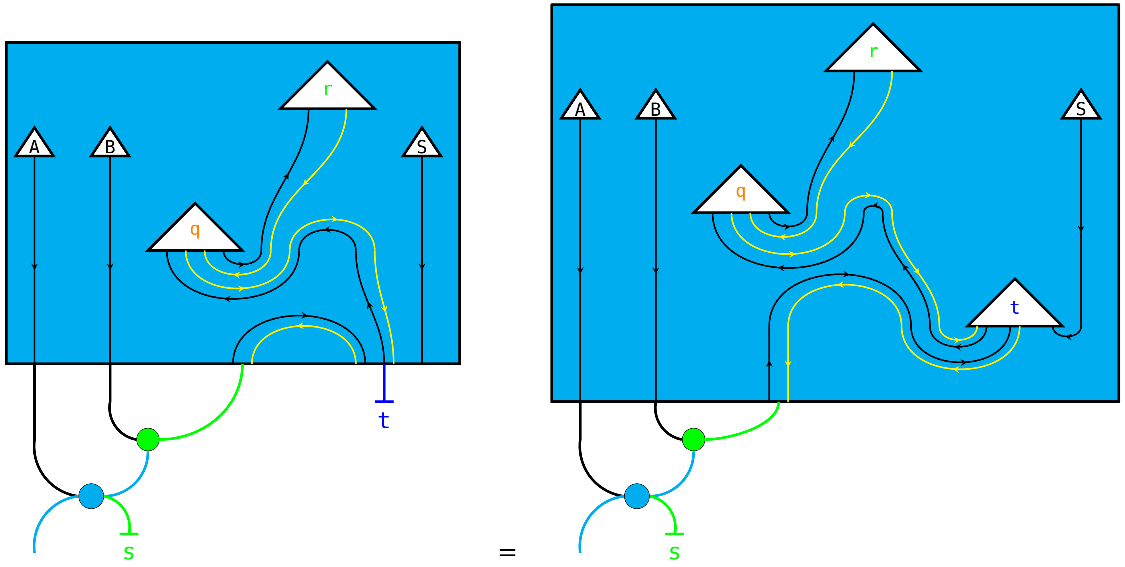
When we pull down the functor-box, the discrete fibration introduces the appropriate choice of diagram from above, corresponding to the intransitive verb case.



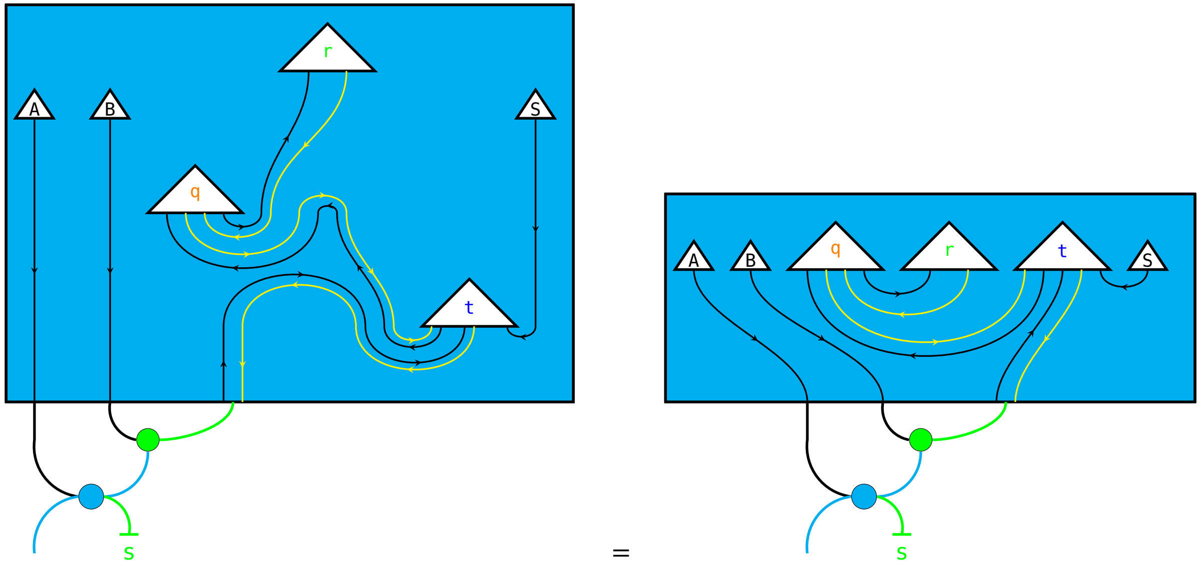
Similarly to `quickly`, we suppose we have a family of processes for the word `to`, one for each noun-arity of verb.



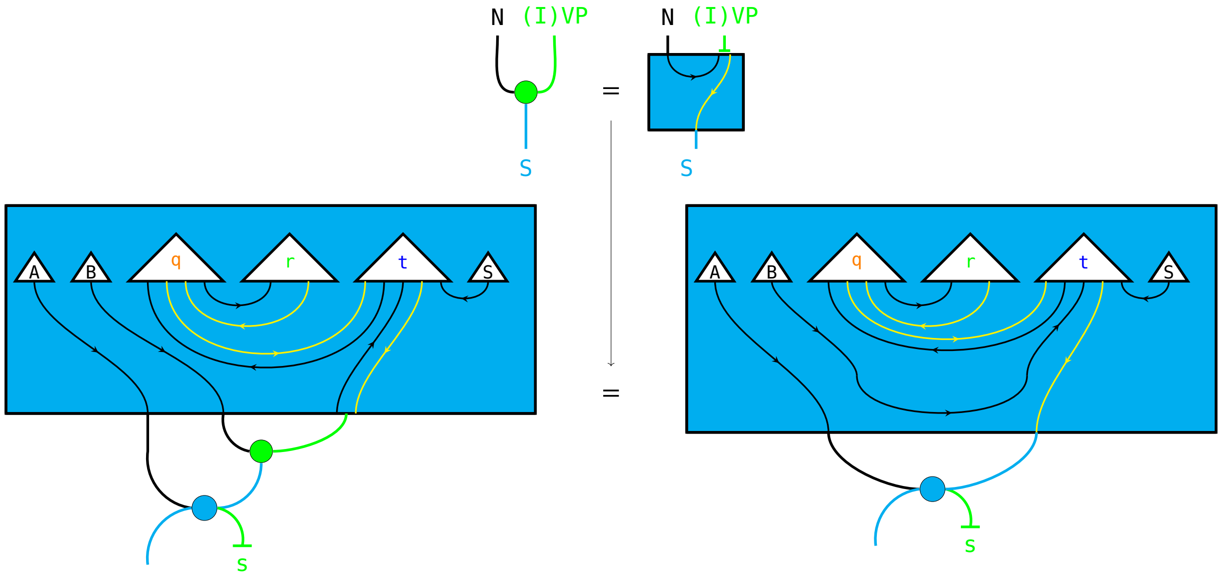
Again the discrete fibration introduces the appropriate choice of `to` when we pull the functor box down.



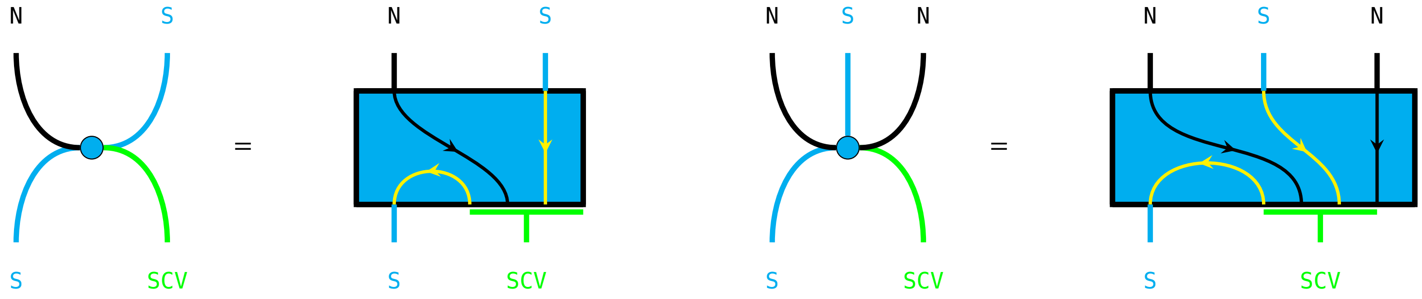
Now we visually simplify the inside of the functor-box by applying yanking equations.



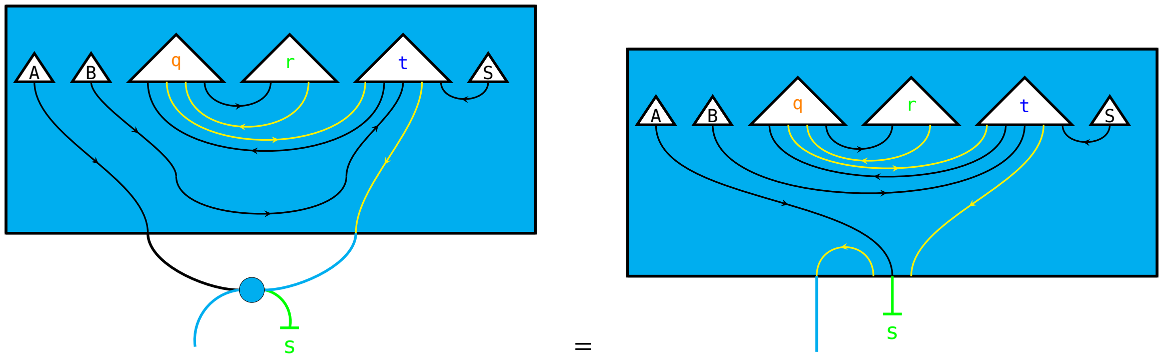
Similarly as before, we can pull the functor-box past the intransitive verb node. There is only one pregroup type $^{-1}n \cdot s$ that corresponds to the grammatical category (I)VP.



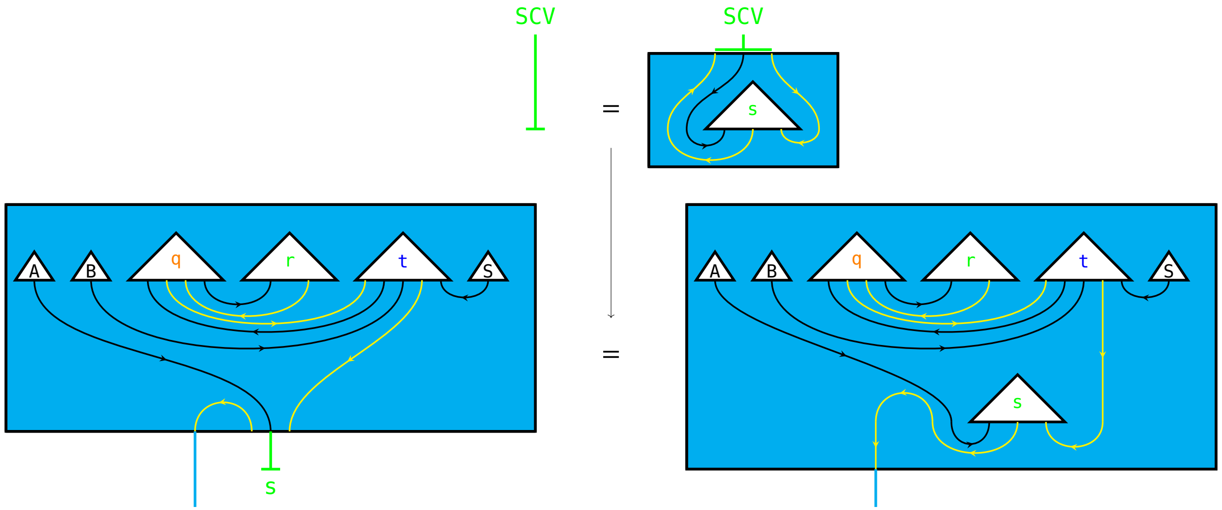
Proceeding similarly, we can pull the functor-box past the sentential-complement-verb node. There are multiple possible pregroup types for SCV, depending on how many noun-phrases are taken as arguments in addition to the sentence. For example, in Alice sees [sentence], sees returns a sentence after taking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



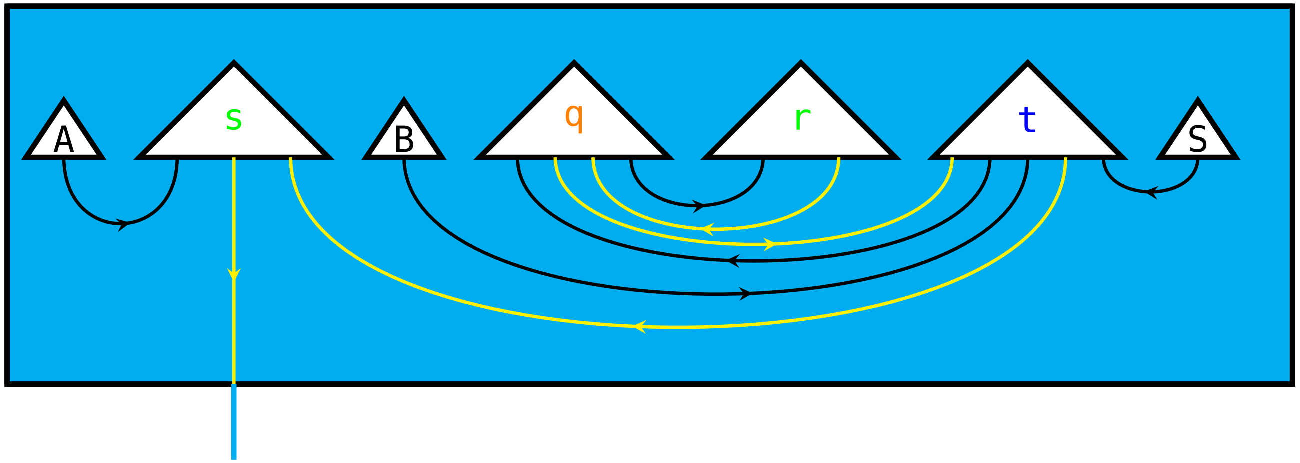
Pulling down the functor box:



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



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



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

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

Remark 2.1.4. Technical addendums.

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

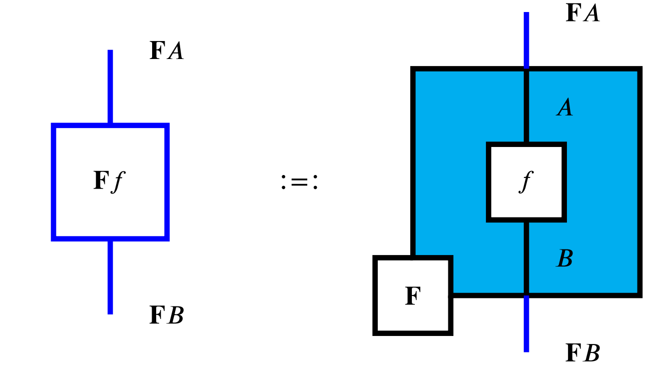
To enforce the well-definedness of the functor $\mathbf{F} : \mathcal{PG} \to \mathcal{G}$ on objects, we may consider the strictified "category of..." [ghicadiagrams]...

2.1.4 *Discrete Monoidal Fibrations*

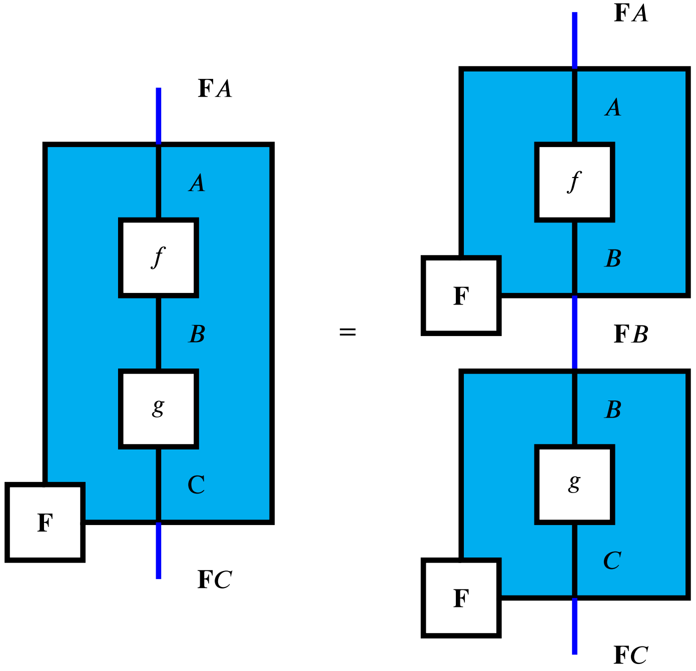
We introduce the concept of a discrete monoidal fibration: a mathematical bookkeeping tool that relates kinds of choices speakers and listerns make when generating and parsing text respectively. We proceed diagrammatically. The first concept is that of a *monoidal functor box*.

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

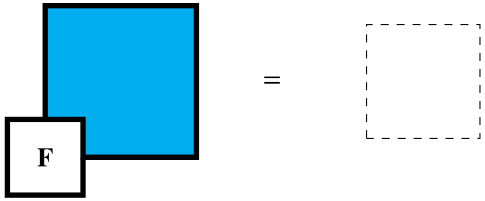
Scholium 2.1.5. Functor boxes are from [meillies]. Expressing the coherence conditions of monoidal functors using equations involving functor boxes as below is not new [meillies]. The idea of a functor being simultaneously monoidal and a fibration is not new [monoidalfibration]. What is new is minor: the express requirement that the lifts of the fibration satisfy interchange, which is in general not guaranteed by just having a functor be monoidal and a (even discrete) fibration [fosco].

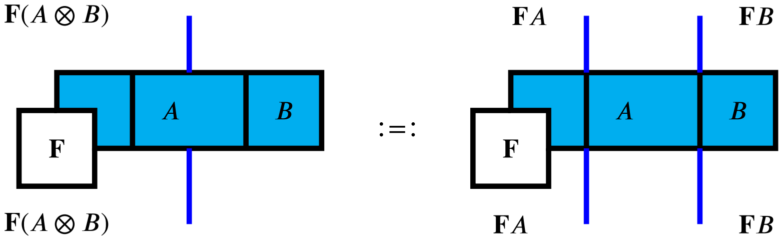


The use of a functor box is like a window from the target category $\mathcal{D}$ into the source category $\mathcal{C}$; when we know that a morphism in $\mathcal{D}$ is the image under $\mathbf{F}$ of some morphism in $\mathcal{C}$, the functor box notation is just a way of presenting all of that data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

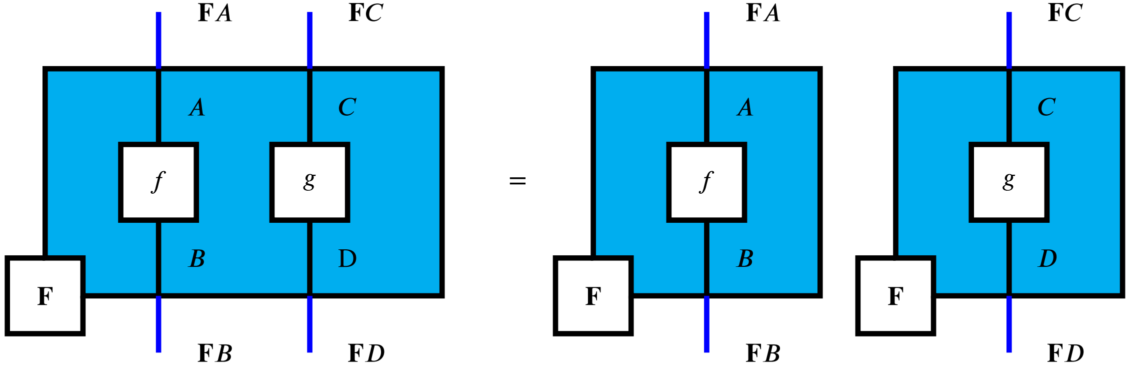


Assume that $\mathbf{F}$ is strict monoidal; without loss of generality by the strictification theorem [], this lets us gloss over the associators and unitors. For $\mathbf{F}$ to be strict monoidal, it has to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A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

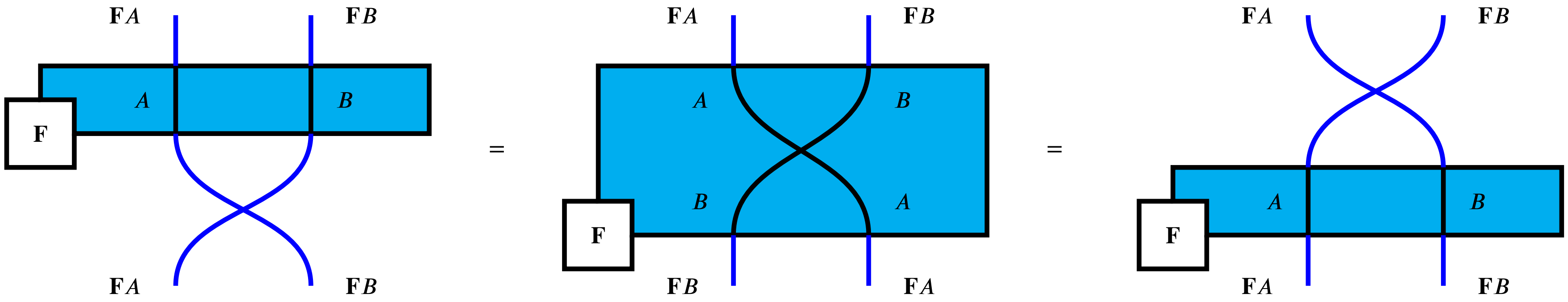




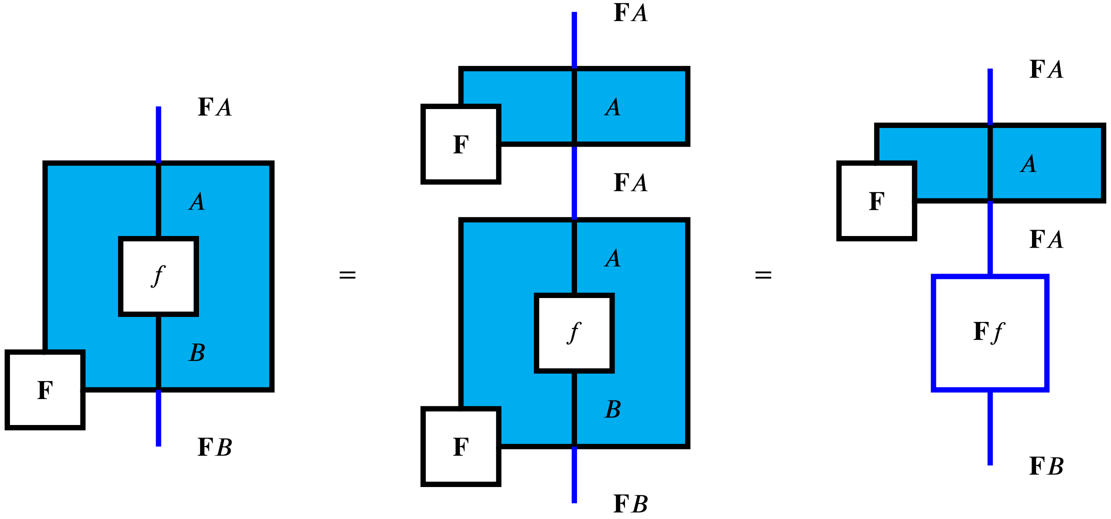
What remains is the monoidality of $\mathbf{F}$, which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes'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



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.



Remark 2.1.6. To motivate fibrations, first observe that by the diagrammatic equations of monoidal categories and functor boxes we have so far, we can always "slide out" the contents of a functor box out of the bottom:



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

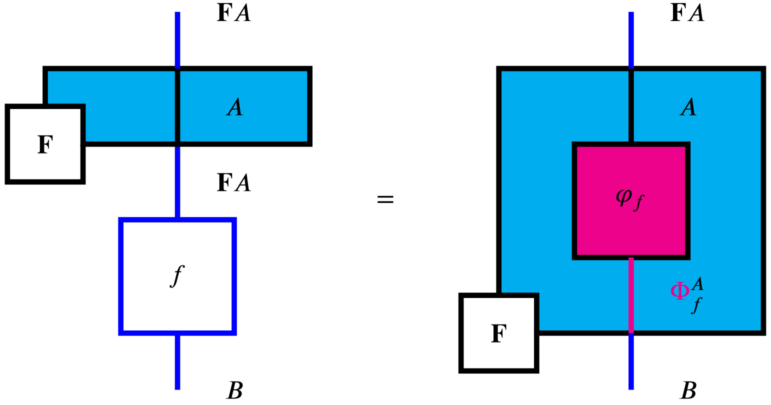
Definition 2.1.7 (Discrete opfibration)**.** $\mathbf{F} : \mathcal{C} \to \mathcal{D}$ is a *discrete fibration* when:

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

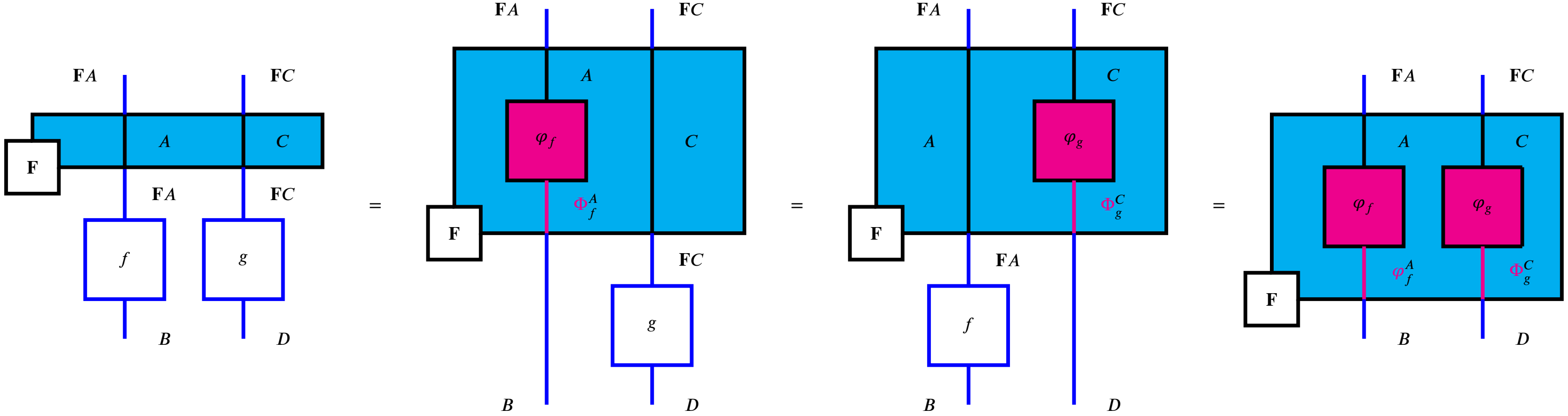
Diagrammatically, we can present all of the above as an equation reminiscent of sliding a morphism *into* a functor box from below.

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

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



Definition 2.1.8 (Monoidal discrete opfibration)**.** We consider **F** to be a *(strict, symmetric) monoidal discrete opfibration* when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the following equations relating lifts to interchange hold:



Remark 2.1.9. 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

2.1.5 Discrete monoidal fibrations for grammatical functions

2.1.6 Extended analysis: Tree Adjoining Grammars

Here is a formal but unenlightening defintion of tree adjoining grammars, which we will convert to diagrams.

Definition 2.1.10 (Tree Adjoining Grammar: Classic Computer Science style)**.** A **TAG** is a tuple $(\mathcal{N}, \mathcal{N}^{\downarrow}, \mathcal{N}^{\star}, \Sigma, \mathcal{I}, \mathcal{A}, \mathbf{s} \in \mathcal{N})$. These denote, respectively:

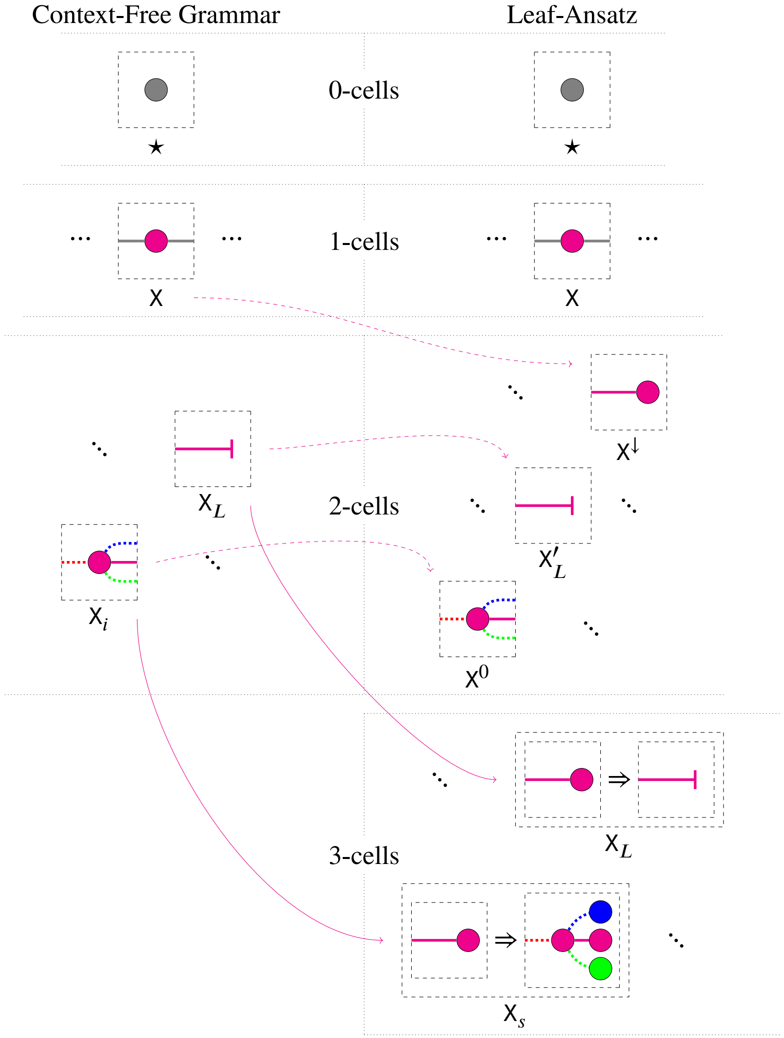
- The *non-terminals*:
  - A set of *non-terminal symbols* $\mathcal{N}$ – these stand in for grammatical types such as $\texttt{NP}$ and $\texttt{VP}$.
  - A bijection $\downarrow : \mathcal{N} \rightarrow \mathcal{N}^{\downarrow}$ which acts as $\texttt{X} \mapsto \texttt{X}^{\downarrow}$. Nonterminals 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 $\star : \mathcal{N} \rightarrow \mathcal{N}^{\star}$ – the same idea as above. This time to mark *foot nodes* on auxiliary trees, which is structural information used by the adjoining operation – discussed later.
- A set of *terminal symbols* Σ – these stand in for the words of the natural language being modelled.
- The *elementary trees*:
  - A set of *initial trees* $\mathcal{I}$, which satisfy the following constraints:
    - * The interior nodes of an initial tree must be labelled with nonterminals from $\mathcal{N}$
    - * The leaf nodes of an initial tree must be labelled from $\Sigma \cup \mathcal{N}^{\downarrow}$
  - A set of *auxiliary trees* $\mathcal{A}$, which satisfy the following constraints:
    - * The interior nodes of an auxiliary tree must be labelled with nonterminals from $\mathcal{N}$
    - * Exactly one leaf node of an auxiliary tree must be labelled with a foot node $\texttt{X}^{\star} \in \mathcal{N}^{\star}$; moreover, this labelled foot node must be the marked counterpart of the root node label of the tree.
    - * All other leaf nodes of an auxiliary tree are labelled from $\Sigma \cup \mathcal{N}^{\downarrow}$

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

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

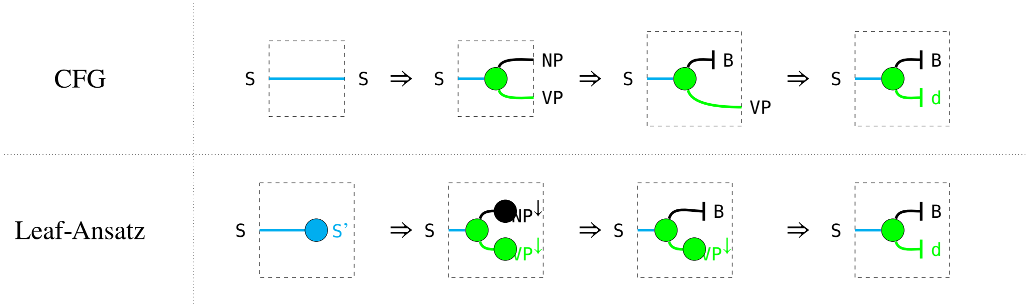
The essence of a tree-*adjoining* grammar is as follows: whereas for a CFG one grows the tree by appending branches and leaves at the top of the tree (substitution), in a TAG one can also sprout subtrees from the middle of a branch (adjoining). Now we show that this gloss is more formal than it sounds,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

Construction 2.1.11 (Leaf-Ansatz of a CFG)**.** Given a signature $\mathfrak{G}$ for a CFG, we construct a new signature $\mathfrak{G}'$ which has the same 0- and 1-cells as $\mathfrak{G}$. See the dashed magenta arrows in the schematic below. For each 1-cell wire type X of $\mathfrak{G}$, we introduce a *leaf-ansatz* 2-cell $\mathsf{X}^{\downarrow}$. For each leaf 2-cell X_L in $\mathfrak{G}$, we introduce a renamed copy X'_L in $\mathfrak{G}'$. Now see the solid magenta arrows in the schematic below. We construct a 3-cell in $\mathfrak{G}'$ for each 2-cell in $\mathfrak{G}$, which has the effect of systematically replacing open output wires in $\mathfrak{G}$ with leaf-ansatzes in $\mathfrak{G}'$.



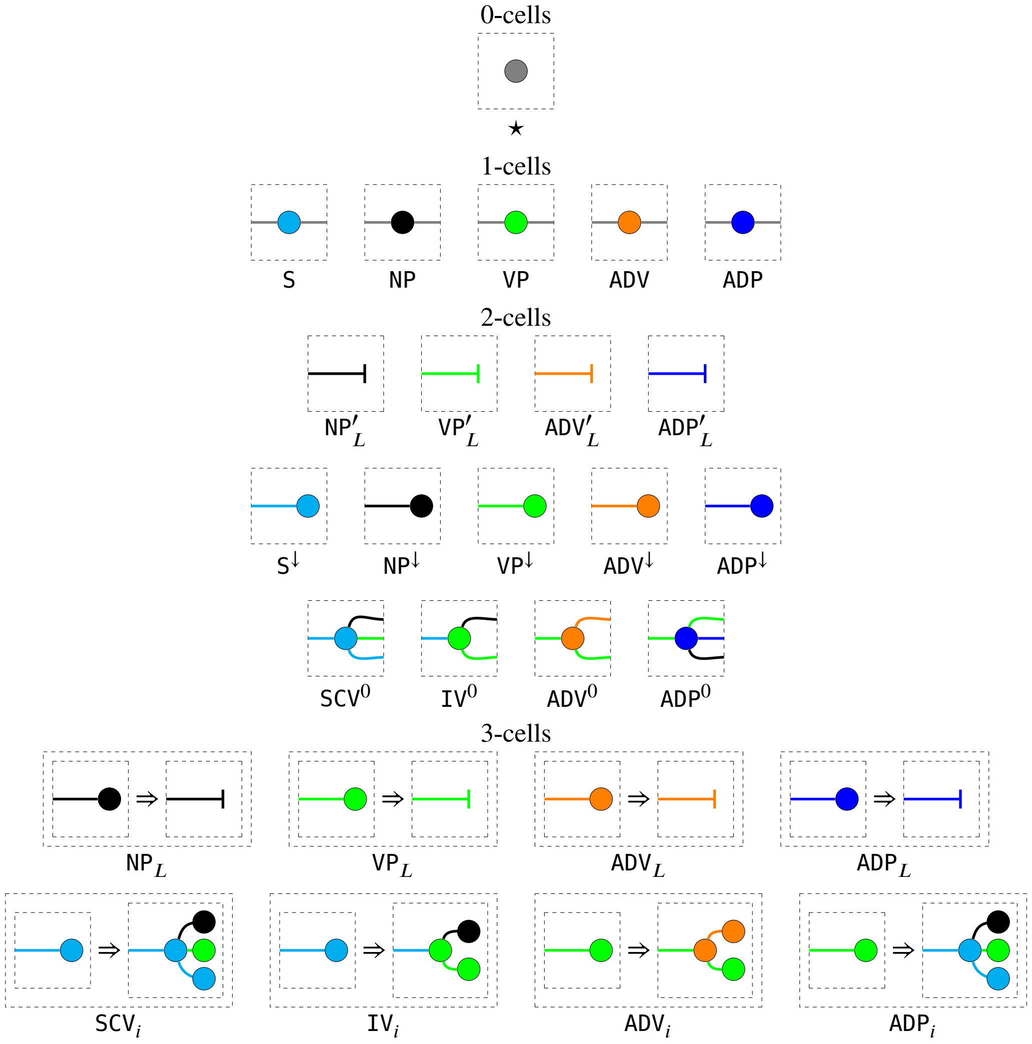
Example 2.1.12. The leaf-ansatz construction just makes formal the following observation: there are multiple equivalent ways of modelling terminal symbols in a rewrite system considered string-diagrammatically. One way (which we have already done) is to treat non-terminals as wires and terminals as effects, so that the presence of an open wire avail-

able for composition visually indicates non-terminality. Another (which is the leaf-ansatz construction) treats all symbols in a rewrite system as leaves, where bookkeeping the distinction between (non-)terminals occurs in the signature. So for a sentence like `Bob drinks`, we have the following derivations that match step for step in the two ways we have considered.



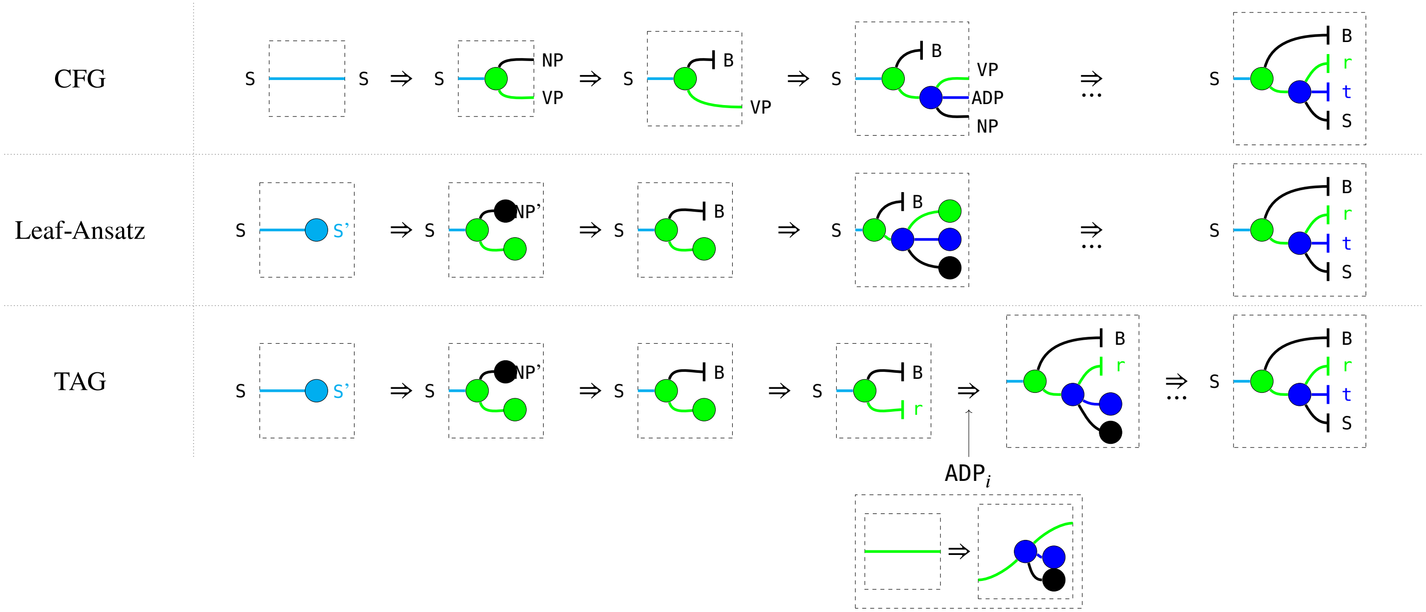
Proposition 2.1.13 (Leaf-ansatzes of CFGs are precisely TAGs with only initial trees and substitution)**.** *Proof.* By construction. Consider a CFG given by 2-categorical signature $\mathfrak{G}$, with leaf-ansatz signature $\mathfrak{G}'$. The types $\mathtt{X}$ of $\mathfrak{G}$ become substitution marked symbols $\mathtt{X}^{\downarrow}$ in $\mathfrak{G}'$. The trees $\mathtt{X}_i$ in $\mathfrak{G}$ become initial trees $\mathtt{X}^0$ in $\mathfrak{G}'$. The 3-cells $\mathtt{X}_s$ of $\mathfrak{G}'$ are precisely substitution operations corresponding to appending the 2-cells $\mathtt{X}_i$ of $\mathfrak{G}$. □

Example 2.1.14 (Leaf-ansatz signature of `Alice sees Bob quickly run to school` CFG)**.**



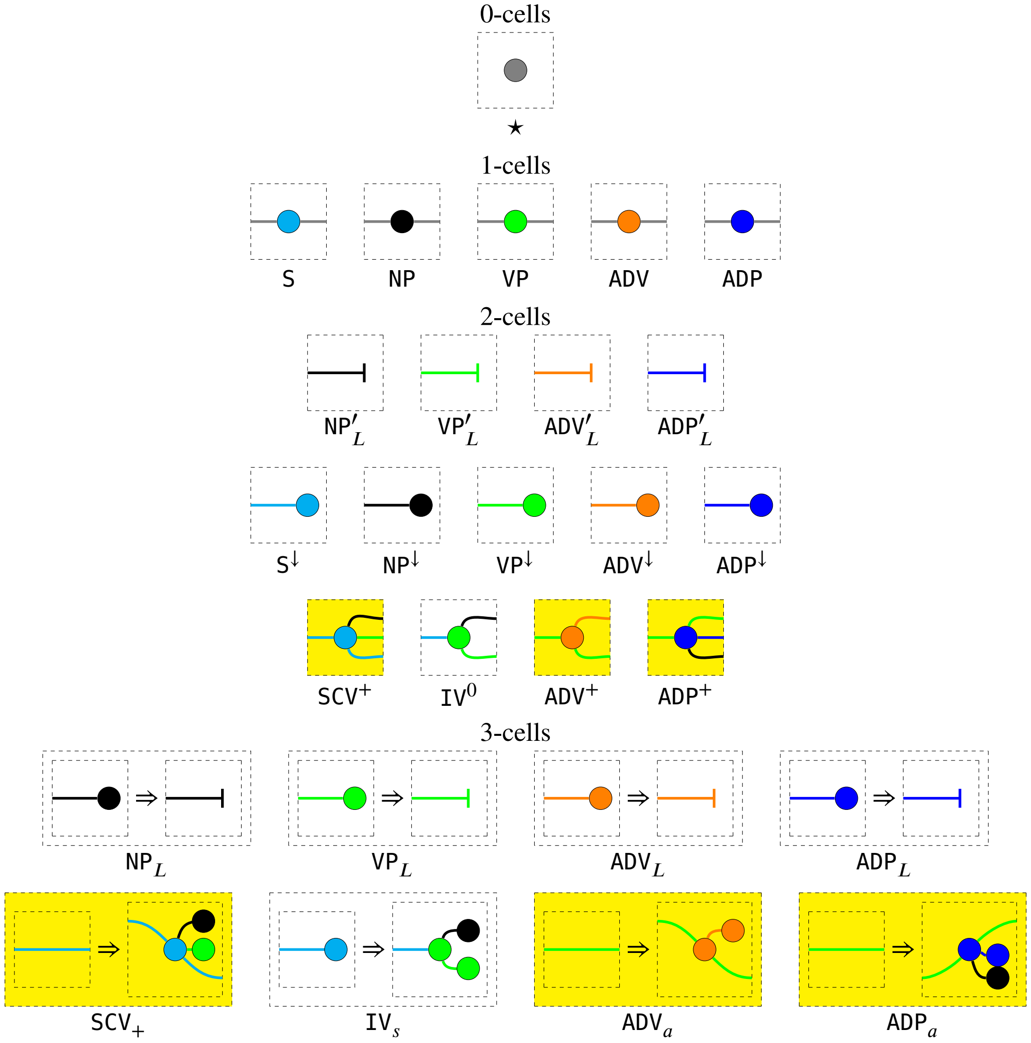
Example 2.1.15 (Adjoining is sprouting subtrees in the middle of branches)**.** One way we might obtain the sentence `Bob runs to school` is to start from the simpler sentence `Bob runs`, and then refine the verb `runs` into `runs to school`. This refinement on part of an already completed sentence is not permitted in CFGs, since terminals can no longer be

modified. The adjoining operation of TAGs gets around this constraint by permitting rewrites in the middle of trees, as follows:



Example 2.1.16 (TAG signature of `Alice sees Bob quickly run to school`)**.** The highlighted 2-cells are auxiliary trees that replace CFG 2-cells for verbs with sentential

complement, adverbs, and adpositions. The highlighted 3-cells are the tree adjoining operations of the auxiliary trees.

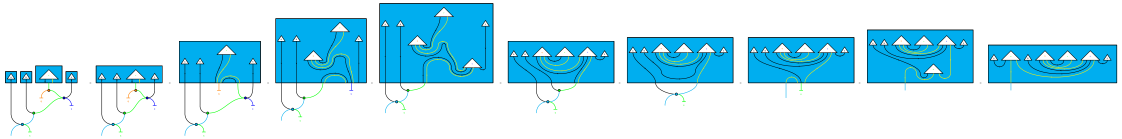


The construction yields as a corollary an alternate proof of Theorem [Joshi 6.1.1...]...

Corollary 2.1.17. For every context-free grammar $\mathfrak{G}$ there exists a tree-adjoining grammar $\mathfrak{G}'$ such that $\mathfrak{G}$ and $\mathfrak{G}'$ are strongly equivalent – both formalisms generate the same set of strings (weak equivalence) and the same abstract syntactic structures (in this case, trees) behind the strings (strong equivalence).

Proof. Proposition 2.1.13 provides one direction of both equivalences. For the other direction, we have to show that each auxiliary tree (a 2-cell) and its adjoining operation (a 3-

cell) in $\mathfrak{G}'$ corresponds to a single 2-cell tree of some CFG signature $\mathfrak{G}$, which we demonstrate by construction. See the example above; the highlighted 3-cells of $\mathfrak{G}'$ are obtained systematically from the auxiliary 2-cells as follows: the root and foot nodes $\mathtt{X}, \mathtt{X}^\star$ indicate which wire-type to take as the identity in the left of the 3-cell, and the right of the 3-cell is obtained by replacing all non-$\mathtt{X}$ open wires $\mathtt{Y}$ with their leaf-ansatzes $\mathtt{Y}^{\downarrow}$. This establishes a correspondence between any 2-cells of $\mathfrak{G}$ considered as auxiliary trees in $\mathfrak{G}'$. □



3

Continuous relations: a palette for toy models

3.1 Continuous Relations

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

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

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

where † denotes the relational converse.

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

Reminder 3.1.1 (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

$$\varnothing, X \in \tau$$

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 $n \in N$:

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

Reminder 3.1.2 (Relational Converse)**.** Recall that a relation $R : S \to T$ is a subset $R \subseteq S \times T$.

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

Reminder 3.1.3 (Continuous function)**.** A function between sets $f : X \to Y$ is a continuous function between topologies $f : (X, \tau) \to (Y, \sigma)$ if

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

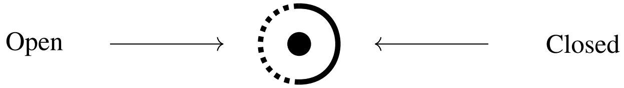
where f^{-1} denotes the inverse image.

3.2 *Continuous Relations by examples*

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

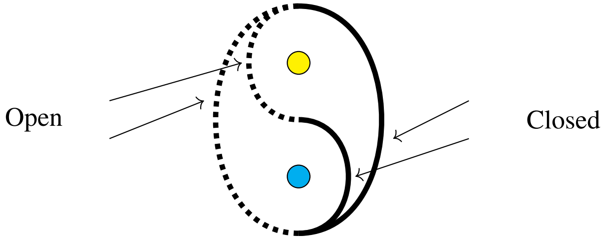
The **singleton space** consists of a single point which is both open and closed. We denote this space $\bullet$. Concretely, the underlying set and topology is

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

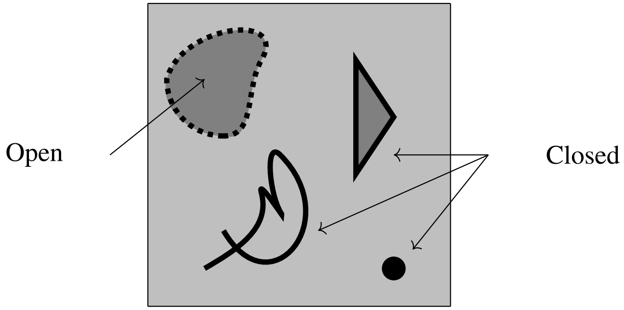


The **Sierpiński space** consists of two points, one of which (in yellow) is open, and the other (in cyan) is closed. We denote this space $\mathcal{S}$. Concretely, the underlying set and topology is:

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



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



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

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

$\mathcal{S} \to \bullet$: There four candidate relations from the Sierpiński space to the singleton, but as we see in Example 3.2.1, not all of them are topological.

Example 3.2.1 (A noncontinuous relation)**.** The relation $\{(0, \bullet)\} \subset \mathcal{S} \times \bullet$ is not a continuous relation: the preimage of the open set $\{\bullet\}$ under this relation is the non-open set $\{0\}$.

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

$\bullet \to$ ■: Proposition 3.2.3 tells us that there are as many continuous relations from the singleton to the unit square as there are subsets of the unit square.

■ $\to \bullet$: Proposition 3.2.4 tells us that there are as many continuous relations from the unit square to the singleton as there are open sets of the unit square.

Terminology 3.2.2. Call a continuous relation $\bullet \to X^\tau$ a **state** of X^τ, and a continuous relation $X^\tau \to \bullet$ a **test** of X^τ.

Proposition 3.2.3. States $R : \bullet \to X^\tau$ correspond with subsets of X.

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

Proposition 3.2.4. Tests $R : X^\tau \to \bullet$ correspond with open sets $U \in \tau$.

Proof. The preimage $R^\dagger(\star)$ of $\star$ must be an open set of X^τ by definition 3.1.4. $R^\dagger(\star)$ is free to specify any open set of X^τ. □

THERE ARE 16 CANDIDATE RELATIONS $\mathcal{S} \to \mathcal{S}$ TO CHECK. A case-by-case approach won't scale, so we could instead identify the building blocks of continuous relations with the same source and target space.

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

THE EMPTY RELATION IS ALWAYS CONTINUOUS.

Proposition 3.2.6. *Proof.* The preimage of the empty relation is always $\varnothing$, which is open by definition. □

Reminder 3.2.5 (Empty relation)**.** The **empty relation** $X \to Y$ relates nothing. It is defined:

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

FULL RELATIONS ARE ALWAYS CONTINUOUS

Proposition 3.2.8. *Proof.* The preimage of any subset of Y – open or not – under the full relation is the whole of X, which is open by definition. □

Reminder 3.2.7 (Full relation)**.** The **full relation** $X \to Y$ relates everything to everything. It is all of $X \times Y$.

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

Proposition 3.2.9. Given an open $U \subseteq X^\tau$, and an arbitrary subset $K \subset Y^\sigma$, the relation $U \times K \subseteq X \times Y$ is open.

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

CONTINUOUS FUNCTIONS ARE ALWAYS CONTINUOUS.

Proposition 3.2.10. If $f : X^\tau \to Y^\sigma$ is a continuous function, then it is also a continuous relation.

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

THE IDENTITY RELATION IS ALWAYS CONTINUOUS. The identity relation is also the "trivial" continuous map from a space to itself, so this also follows from Proposition 3.2.10.

Reminder 3.2.11 (Identity relation)**.** The **identity relation** $X \to X$ relates anything to itself. It is defined:

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

Proposition 3.2.12. *Proof.* The preimage of any open set under the identity relation is itself, which is open by assumption. □

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

Reminder 3.2.13 (Union, intersection, and ordering of relations)**.** Recall that relations $X \to Y$ can be viewed as subsets of $X \times Y$. So it makes sense to speak of the union and intersection of relations, and of partially ordering them by inclusion.

Proposition 3.2.14. If $R, S : X^\tau \to Y^\sigma$ are continuous relations, so are $R \cap S$ and $R \cup S$.

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

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

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

Corollary 3.2.15. Continuous relations $X^\tau \to Y^\sigma$ are closed under arbitrary union and finite intersection. Hence, continuous relations $X^\tau \to Y^\sigma$ form a topological space where each continuous relation is an open set on the base space $X \times Y$, where the full relation $X \to Y$ is "everything", and the empty relation is "nothing".

A TOPOLOGICAL BASIS FOR SPACES OF CONTINUOUS RELATIONS

Reminder 3.2.16 (Topological Basis)**.** $\mathfrak{b} \subseteq \tau$ is a basis of the topology τ if every $U \in \tau$ is expressible as a union of elements of $\mathfrak{b}$. Every topology has a basis (itself). Minimal bases are not necessarily unique.

Definition 3.2.17 (Partial Functions)**.** A **partial function** $X \to Y$ is a relation for which each $x \in X$ has at most a single element in its image. In particular, all functions are special cases of partial functions, as is the empty relation.

Lemma 3.2.18 (Partial functions are a $\cap$-ideal)**.** The intersection $f \cap R$ of a partial function $f : X \to Y$ with any other relation $R : X \to Y$ is again a partial function.

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

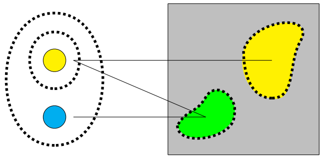


Figure 3.1: Regions of ■ in the image of the yellow point alone will be coloured yellow, and regions in the image of both yellow and cyan will be coloured green:

Lemma 3.2.19 (Any single edge can be extended to a continuous partial function)**.** Given any $(x, y) \in X \times Y$, there exists a continuous partial function $X^\tau \to Y^\tau$ that contains (x, y).

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

Proposition 3.2.20. Continuous partial functions form a topological basis for the space $(X \times Y)^{(\tau \multimap \sigma)}$, where the opens are continuous relations $X^\tau \to Y^\sigma$.

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

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

The $\supseteq$ direction is evident, while the $\subseteq$ direction follows from Lemma 3.2.19. By Lemma 3.2.18, every $R \cap F$ term is a partial function, and by Corollary 3.2.15, continuous. □

$\mathcal{S} \to \mathcal{S}$: We can use Proposition 3.2.20 to write out the topological basis of continuous partial functions, from which we can take unions to find all the continuous relations, which we depict in Figure 3.5.

$\mathcal{S} \to \blacksquare$: Now we use the colour convention of the points in $\mathcal{S}$ to "paint" continuous relations on the unit square "canvas", as in Figures 3.1 and 3.2. So each continuous relation is a painting, and we can characterise the paintings that correspond to continuous relations $\mathcal{S} \to \blacksquare$ in words as follows: Cyan only in points and lines, and either contained in or at the boundary of yellow or green. Have as much yellow and green as you like.

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

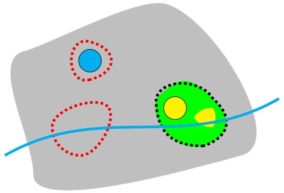


Figure 3.2: Regions in the image of the cyan point alone cannot be open sets by continuity, so they are either points or lines. Points and lines in cyan must be surrounded by an open region in either yellow or green, or else we violate continuity (open sets in red).

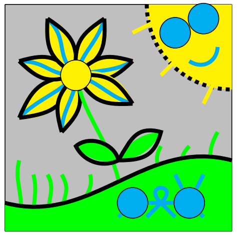


Figure 3.3: A continuous relation $\mathcal{S} \to \blacksquare$: "Flower and critter in a sunny field".

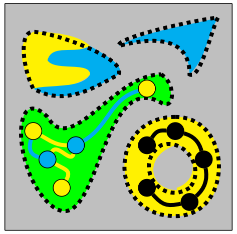


Figure 3.4: A continuous relation $\blacksquare \to \mathcal{S}$: "still math?". Black lines and dots indicate gaps.

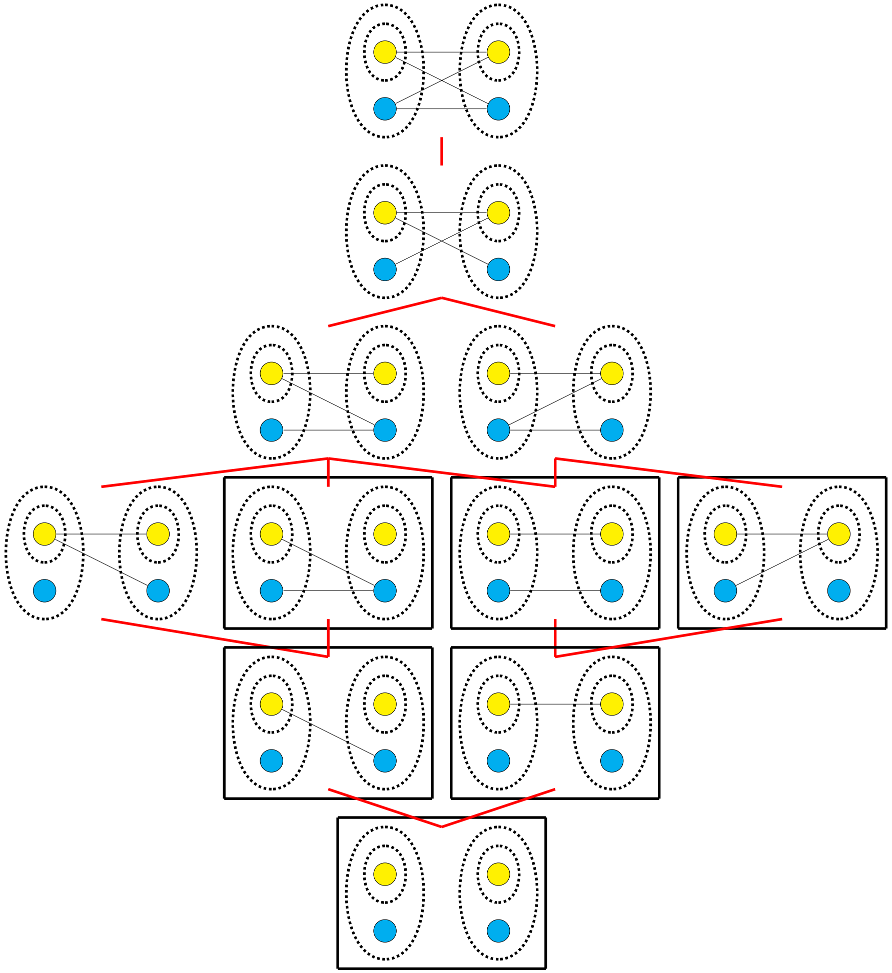


Figure 3.5: Hasse diagram of all continuous relations from the Sierpiński space to itself. Each relation is depicted left to right, and inclusion order is bottom-to-top. Relations that form the topological basis are boxed.

ONE MORE EXAMPLE FOR FUN: $[0,1] \to \blacksquare$: We know how continuous functions from the unit line into the unit square look.

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

ANY PAINTING IS A CONTINUOUS RELATION $[0,1] \to \blacksquare$. By colour-coding $[0,1]$ and controlling brushstrokes, we can do quite a lot. Now we would like to develop the abstract machinery required to *formally* paint pictures with words.

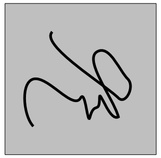


Figure 3.6: continuous functions $[0,1] \to \blacksquare$ follow the naïve notion of continuity: `"a line one can draw on paper without lifting the pen off the page"`.

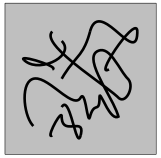


Figure 3.7: So a continuous partial function is `"(countably) many (open-ended) lines, each of which one can draw on paper without lifting the pen off the page."`

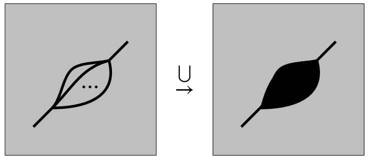


Figure 3.8: We can control the thickness of the brushstroke, by taking the union of (uncountably) many lines.

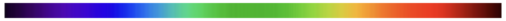


Figure 3.9: Assign the visible spectrum of light to $[0,1]$. Colour open sets according to perceptual addition of light, computing brightness by normalising the measure of the open set.

Figure 3.10: Like it or not, a continuous relation $[0,1] \to \blacksquare$: "The Starry Night", by Vincent van Gogh.

3.3 The category **TopRel**

Proposition 3.3.1. continuous relations form a category **TopRel**.

Proof. IDENTITIES: Identity relations, which are always topological.

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

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

ASSOCIATIVITY OF COMPOSITION: Inherited from **Rel**. □

3.4 Biproducts

We exhibit a free-forgetful adjunction between **Rel** and **TopRel**.

Definition 3.4.1 (F: **Rel** → **TopRel**)**.** We define the action of the functor F:

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

On morphisms $F(X \overset{R}{\rightarrow} Y) := X^\star \overset{R}{\rightarrow} Y^\star$. Recall that any relation between sets is continuous with respect to the discrete topology.

Evidently identities and associativity of composition are preserved.

Definition 3.4.2 (U: **TopRel** → **Rel**)**.** We define the action of the functor U:

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

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

Evidently identities and associativity of composition are preserved.

Proposition 3.4.3 ($F \dashv U \dashv F$)**.** *Proof.* By triangular identities.

The composite FU is precisely equal to the identity functor on **Rel**. The unit natural transformation $1_{\mathbf{Rel}} \Rightarrow FU$ we take to be the identity morphisms.

$$\eta_X := \mathrm{id}_X$$

The counit natural transformation $UF \Rightarrow 1_{\mathbf{TopRel}}$ we define:

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

Now we verify the triangle identities...

$$placeholder$$

□

Corollary 3.4.4. **TopRel** has a zero object and biproducts.

Proof. □

3.5 Symmetric Monoidal Closed structure

Reminder 3.5.1 (Product Topology)**.** We denote the product topology of X^τ and Y^σ as $(X \times Y)^{(\tau \times \sigma)}$. $\tau \times \sigma$ is the topology on $X \times Y$ generated by the basis $\{t \times s : t \in \mathfrak{b}_\tau, s \in \mathfrak{b}_\sigma\}$, where $\mathfrak{b}_\tau$ and $\mathfrak{b}_\sigma$ are bases for τ and σ respectively.

Proposition 3.5.2. ($\mathbf{TopRel}, \{\star\}^\bullet, X^\tau \otimes Y^\sigma := (X \times Y)^{(\tau \times \sigma)}$) is a symmetric monoidal closed category.

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

Reminder 3.5.3 (Product of relations)**.** For relations between sets $R : X \to Y, S : A \to B$, the product relation $R \times S : X \times A \to Y \times B$ is defined to be

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

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

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

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

Unitors: The left unitors $\lambda_{X^\tau} : \bullet \times X^\tau \to X^\tau$ maps $(\star, x) \mapsto x$, and we reverse the direction of the mapping to obtain the inverse $\lambda^{-1}_{X^\tau}$. The construction is symmetric for the right unitors ρ_{X^τ}.

Associators:

The associators $\alpha_{X^\tau, Y^\sigma, Z^\rho}$

Braids:

...

Coherences:

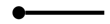
...

□

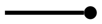
3.6 **TopRel** diagrammatically

3.6.1 Relations that are always continuous

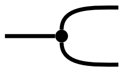
Here are five continuous relations for any X^τ:



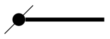
everything



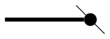
delete



copy

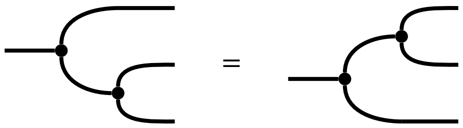


nothing (state)

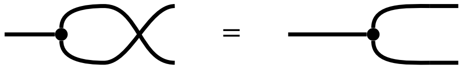


nothing (test)

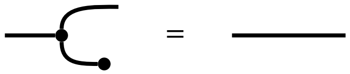
Copy and delete obey the following equalities:



coassociativity

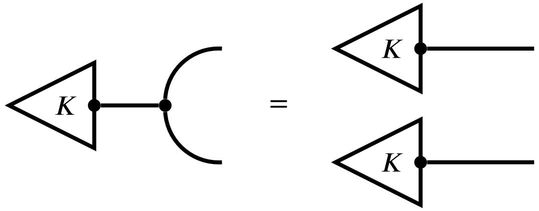


cocommutativity

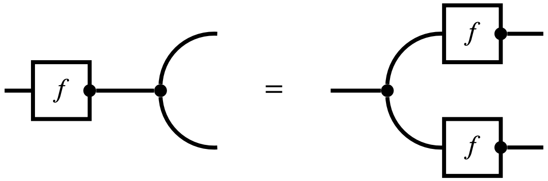


counitality

The copy map can also be used to distinguish the deterministic maps – points and functions – which we notate with an extra dot.

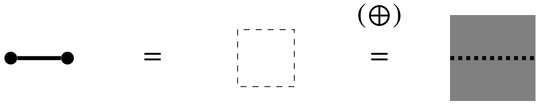


point

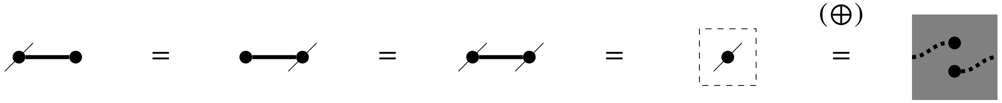


function

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

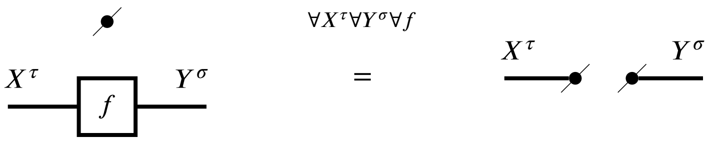


(b)one



zero

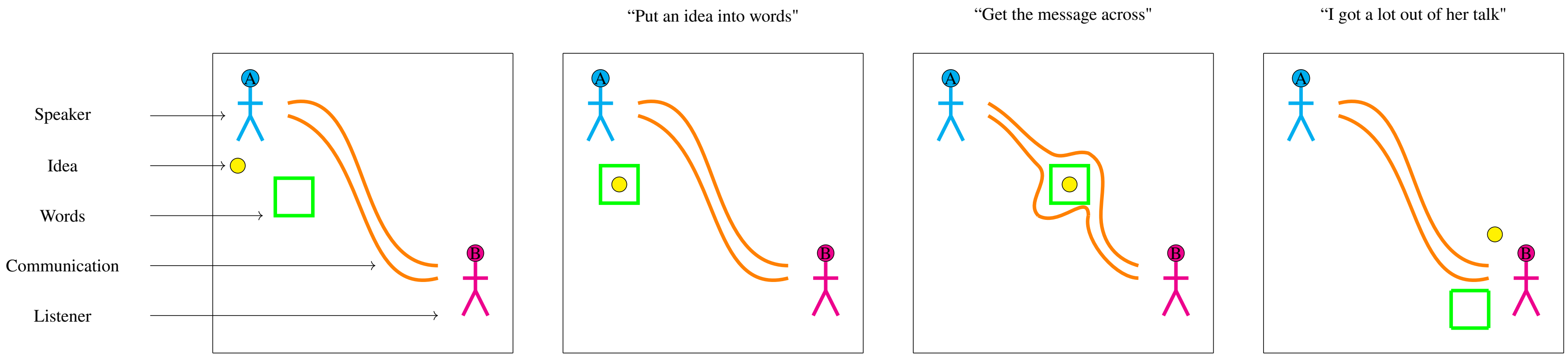
Zero scalars turn entire diagrams into zero morphisms. There is a zero-morphism for every input-output pair of objects in **TopRel**.


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

zero

3.7 *Populating space with shapes using sticky spiders*

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



If you are already happy to treat such doodles as formal, then I think you're alright, please skip the rest of this chapter. If you are a category theorist or just curious, hello, how are you, please do write me to share your thoughts if you care to, and please skip the rest of this paragraph. If you are still reading, I assume you are some kind of smelly epistemic-paranoiac Bourbaki-thrall sets-and-lambdas math-phallus-worshipping truth-condition-blinded symbol-pusher who takes things too seriously. I want you to know that I feel pity and disdain for you with what mild strength I can muster to care about you, and I promise you my feelings towards you are warmer than the oblivion of irrelevance that awaits us all. I hope you choke on the mathematics in this chapter. I will begin the intimidation immediately.

We provide a generalisation (Definition 3.7.5) of special commutative frobenius algebras in **TopRel** that cohere with idempotents in the category. The relation of (†)-special commutative algebras in **FdHilb** to model observables in quantum mechanics is well-studied [], as is the role of idempotents in generalisations of quantum logic to arbitrary categories [], therefore this generalisation may be viewed as a unification of these ideas to define doodles in **TopRel**. N.B. we are not quite taking the Karoubi envelope of **TopRel**, as we are restricting the idempotents we wish to consider to only those that also behave appropriately as observables. The reason for this restriction lies in Theorem 3.7.8 which provides a diagrammatic characterisation result that allows us to precisely identify when any idempotent in the category splits though a discrete topology. The discrete topology thus acts as a set of labels, where the (pre)images of each element under section and retract behave as the kind of shapes we wish to consider. As an interlude, we demonstrate how we may construct families of such idempotent-coherent special commutative frobenius algebras on R^2 to provide a monoidal generalisation (c.f. the monoidal computer framework in []) of **FinRel** equipped with a Turing object [], thus satisfying Justification **??**. Then we proceed to define configuration spaces of collections of shapes up to rigid displacement, and we develop a relational analogue of homotopy to model motions as paths in configuration space, along with appropriate extensions to accommodate nonrigid phenomena. If you are still reading and not already a category theorist nor just curious, I will drop the jargon now, because you are probably either questioning or deeply committed to your false ideals, both of which I respect, but just to spite the latter, I will proceed to do everything diagrammatically as much as possible.

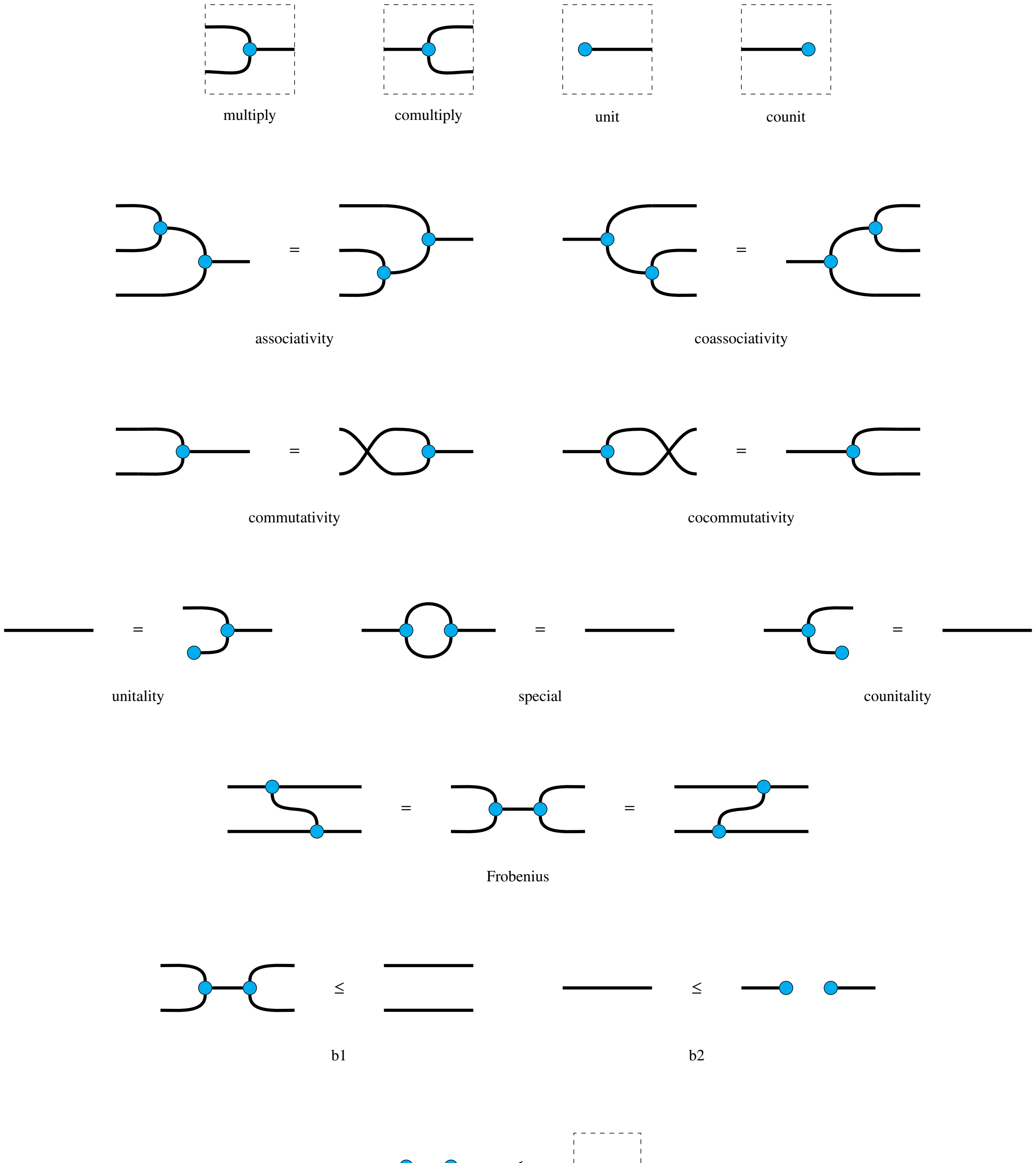


Figure 3.11: The generators (in dashed boxes) and relations that make a spider. When the spider satisfies in addition the three inequalities b1-3, we call it a **relation-spider**.

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

Example 3.7.2 (The copy-compare spiders of **Rel** are not always continuous)**.** The compare map for the Sierpiński space is not continuous, because the preimage of $\{0, 1\}$ is $\{(0, 0), (1, 1)\}$, which is not open in the product space of $\mathcal{S}$ with itself.

Reminder 3.7.1 (copy-compare spiders of **Rel**)**.** For a set X, the *copy* map $X \to X \times X$ is defined:

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

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

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

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

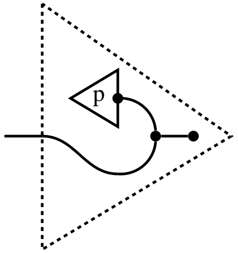
$$\{(x, \star) : x \in X\}$$

and *everything*:

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

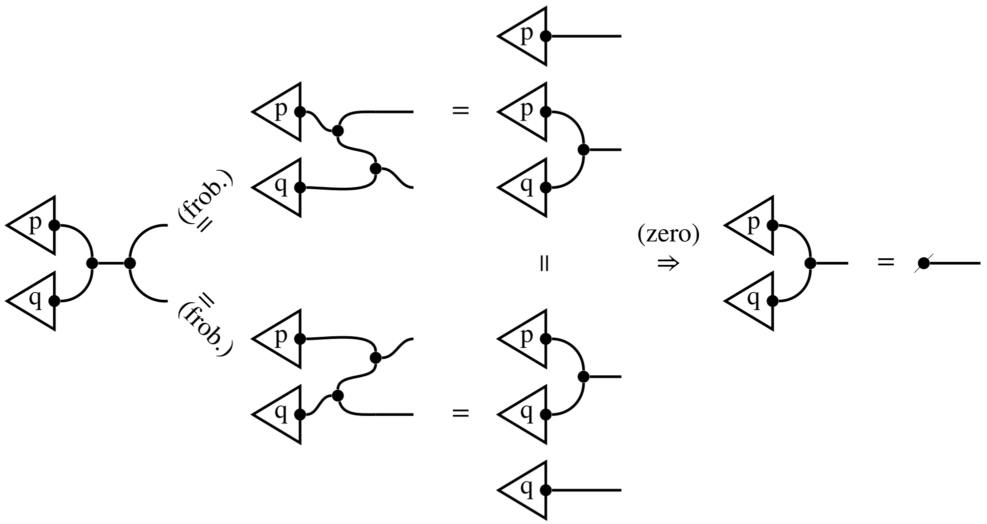
Proposition 3.7.3. The copy map is a spider iff the topology is discrete.

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

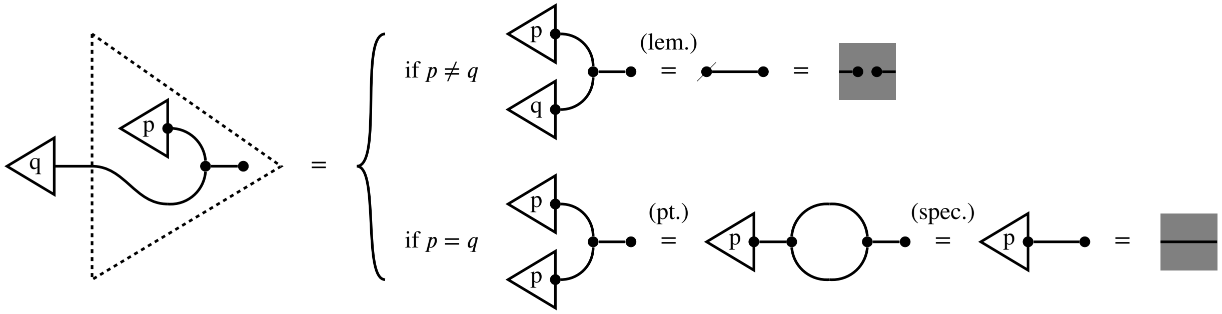


It will suffice to show that this open set is the singleton $\{p\}$ – when all singletons are open, the topology is discrete. As a lemma, using frobenius rules and the property of zero morphisms, we can show that comparing distinct points

$p \neq q$ yields the $\varnothing$ state.



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



□

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

We can use split idempotents to transform copy-spiders from discrete topologies to almost-spiders on other spaces. We can graphically express the behaviour of a split idempotent e as follows, where the semicircles for the section and retract r, s form a visual pun.

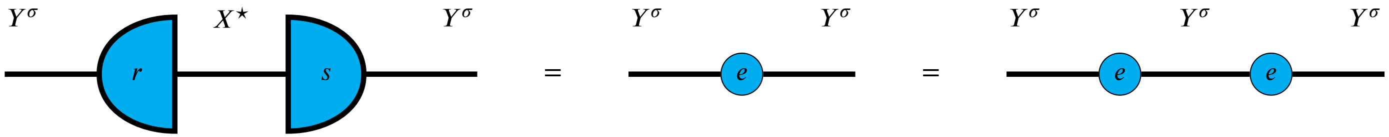
Reminder 3.7.4 (Split idempotents)**.** An **idempotent** in a category is a map $e : A \to A$ such that

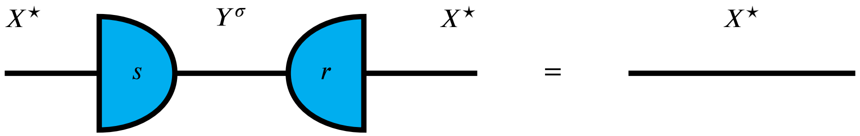
$$A \xrightarrow{e} A \xrightarrow{e} A = A \xrightarrow{e} A$$

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

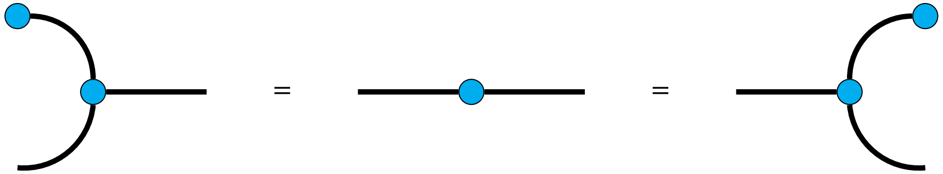
$$A \xrightarrow{e} A = A \xrightarrow{r} B \xrightarrow{s} A$$

$$B \xrightarrow{s} A \xrightarrow{r} B = B \xrightarrow{id} B$$

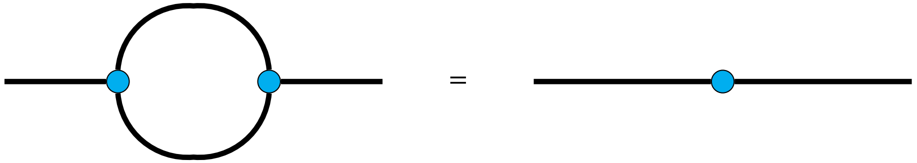




Definition 3.7.5 (Sticky spiders)**.** A **sticky spider** (or just an *e*-spider, if we know that *e* is a split idempotent), is a spider *except* every identity wire on any side of an equation in Figure 3.11 is replaced by the idempotent *e*.

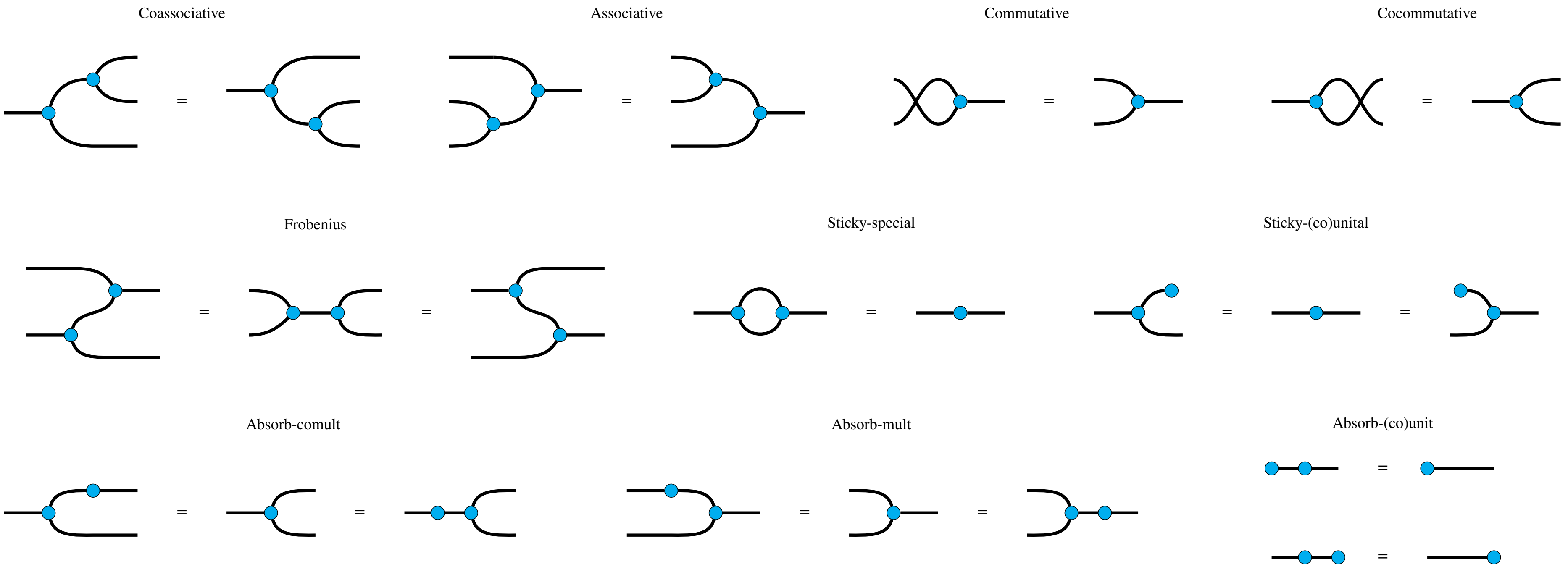


e-(co)unitality



e-special

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



Coassociative

Associative

Commutative

Cocommutative

Frobenius

Sticky-special

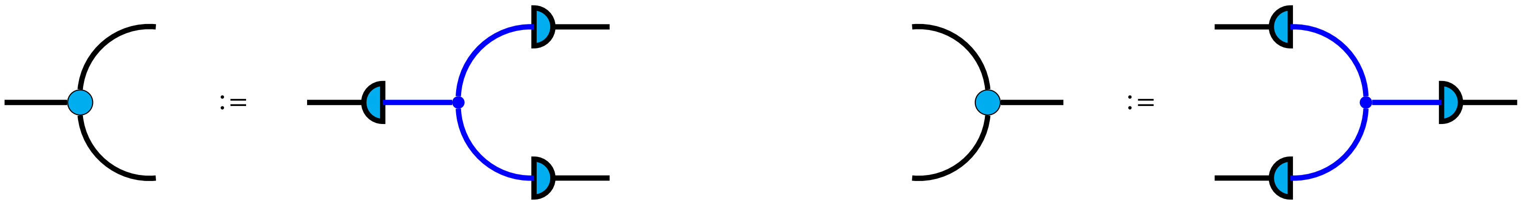
Sticky-(co)unital

Absorb-comult

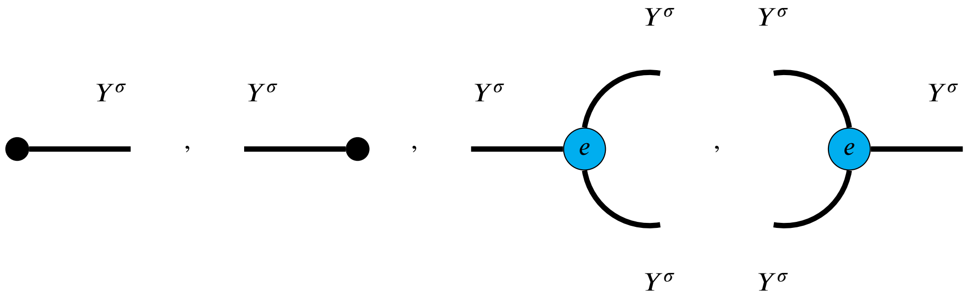
Absorb-mult

Absorb-(co)unit

Construction 3.7.6 (Sticky spiders from split idempotents)**.** Given an idempotent $e : Y^\sigma \to Y^\sigma$ that splits through a discrete topology $X^\star$, we construct a new (co)multiplication as follows:

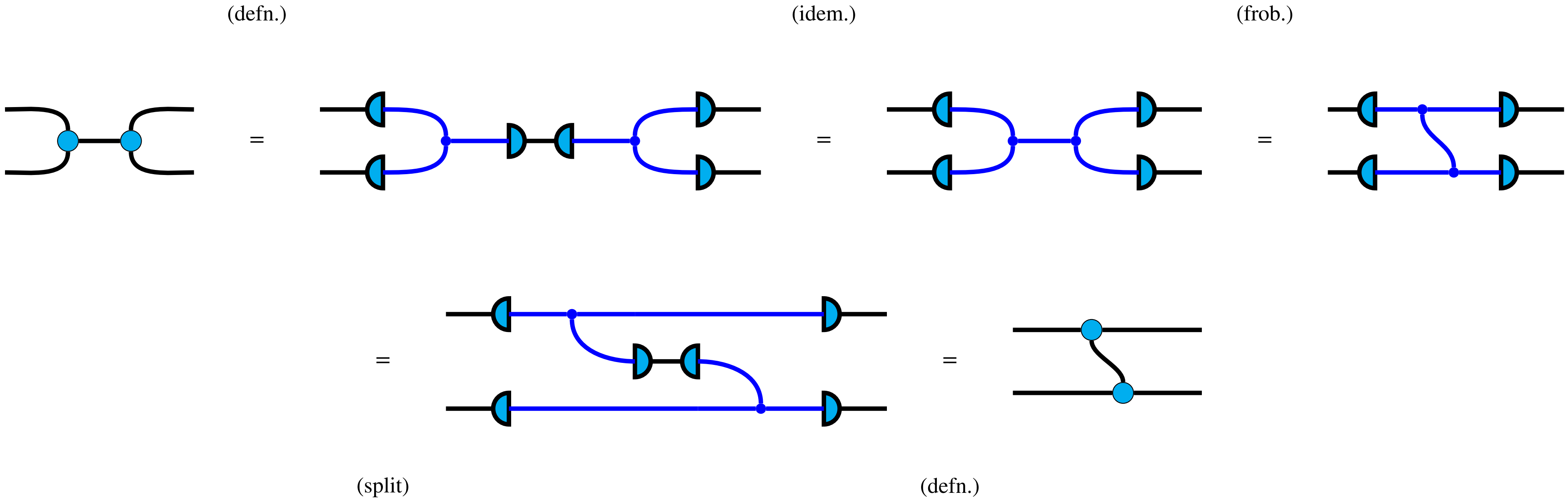


Proposition 3.7.7 (Every idempotent that splits through a discrete topology gives a sticky spider)**.**



is a sticky spider

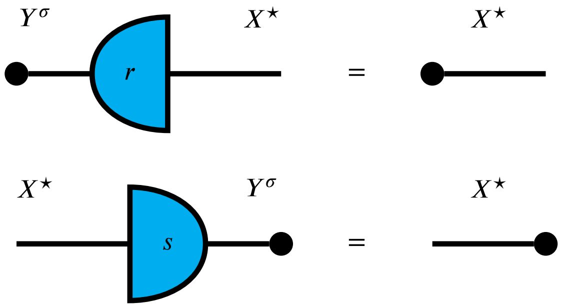
Proof. We can check that our construction satisfies the frobenius rules as follows. We only present one equality; the rest follow the same idea.



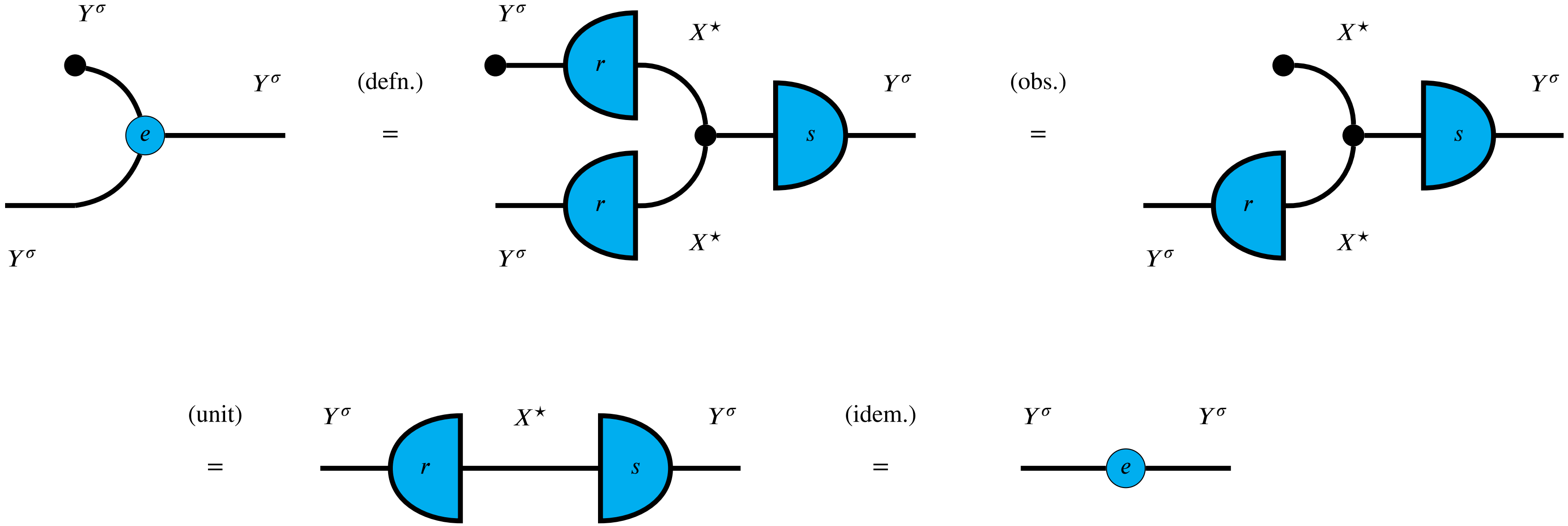
To verify the sticky spider rules, we first observe that since

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

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



Now we show that e-unitality holds:

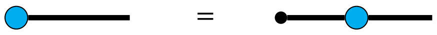


The proofs of e-counitality, and e-speciality follow similarly. □

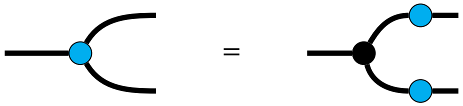
We can prove a partial converse of Proposition 3.7.7: we can identify two diagrammatic equations that tell us precisely when a sticky spider has an idempotent that splits though some discrete topology.

Theorem 3.7.8. 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.

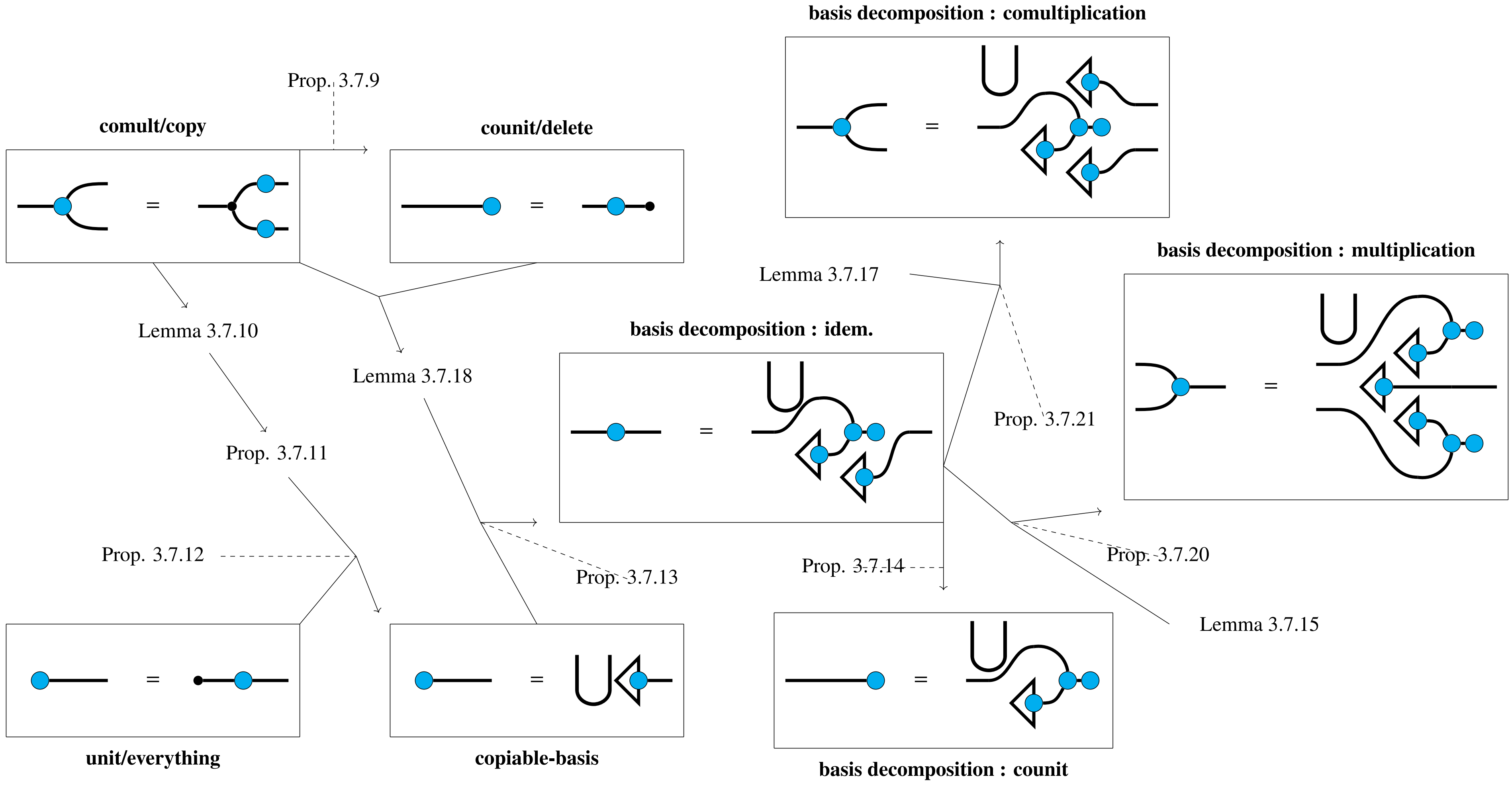
Unit/everything



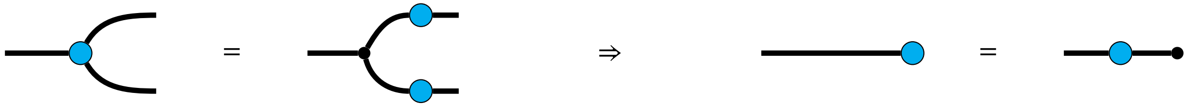
Comult/copy



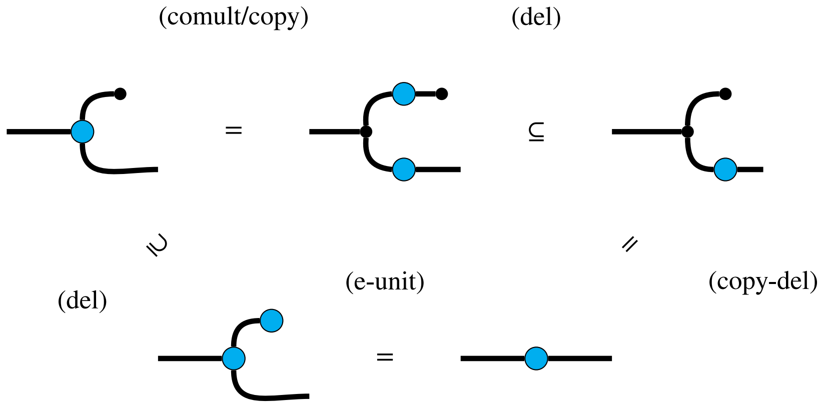
The proof is rather involved, so we provide a map below of the various lemmas and propositions that will yield the claim.


basis decomposition : comultiplication
Prop. 3.7.9
comult/copy
counit/delete
basis decomposition : multiplication
Lemma 3.7.17
Lemma 3.7.10
basis decomposition : idem.
Lemma 3.7.18
Prop. 3.7.21
Prop. 3.7.11
Prop. 3.7.12
Prop. 3.7.13
Prop. 3.7.14
Prop. 3.7.20
Lemma 3.7.15
unit/everything
copiable-basis
basis decomposition : counit
=

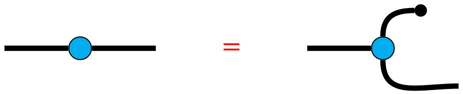
Proposition 3.7.9 (comult/copy implies counit/delete)**.**



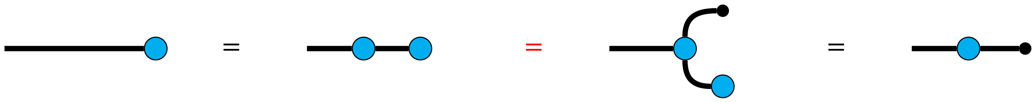
Proof.



So:

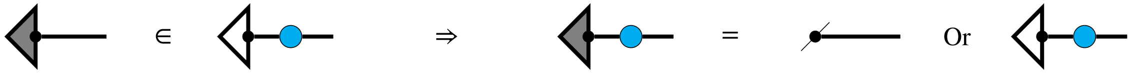


So:



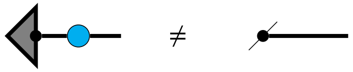
□

Lemma 3.7.10 (All-or-Nothing)**.** Consider the set $e(\{x\})$ obtained by applying the idempotent e to a singleton $\{x\}$, and take an arbitrary element $y \in e(x)$ of this set. Then $e(\{y\}) = \varnothing$ or $e(\{x\}) = e(\{y\})$. Diagrammatically:

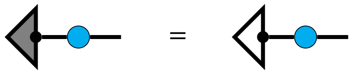


Proof.

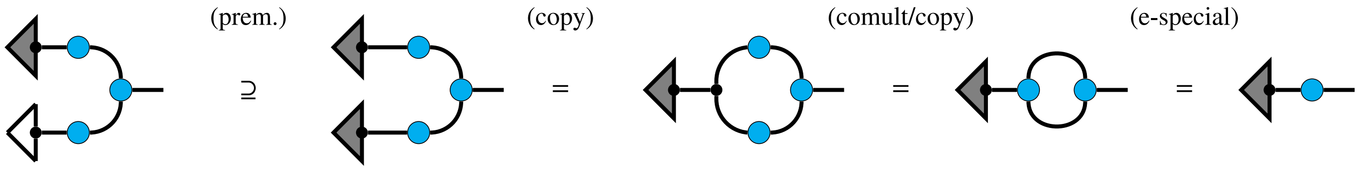
Suppose



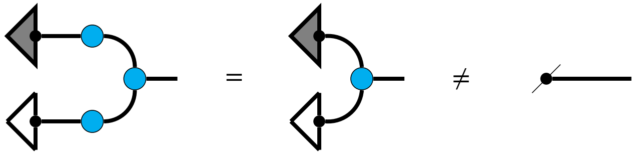
For the claim, we seek:



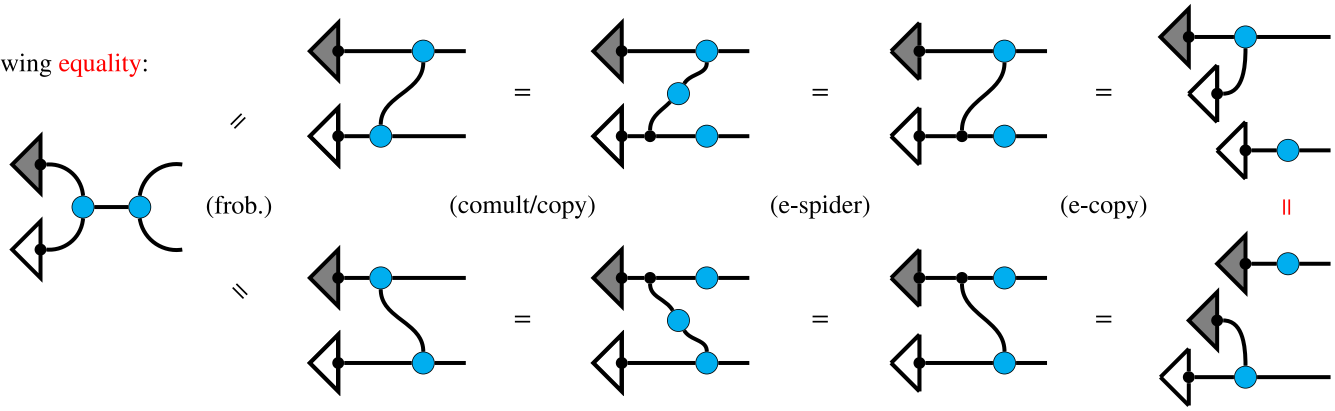
We have the following inclusion:



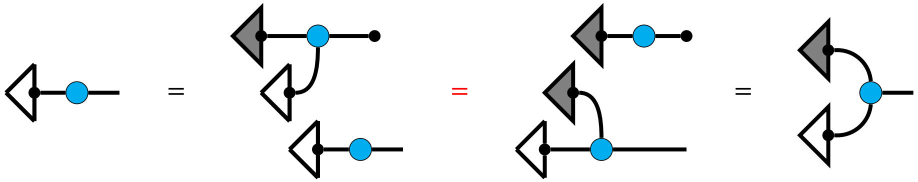
Therefore:



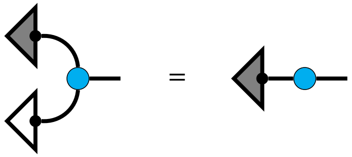
So we have the following equality:



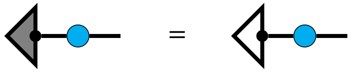
Which implies:



and symmetrically,

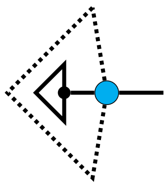


So we have the claim:

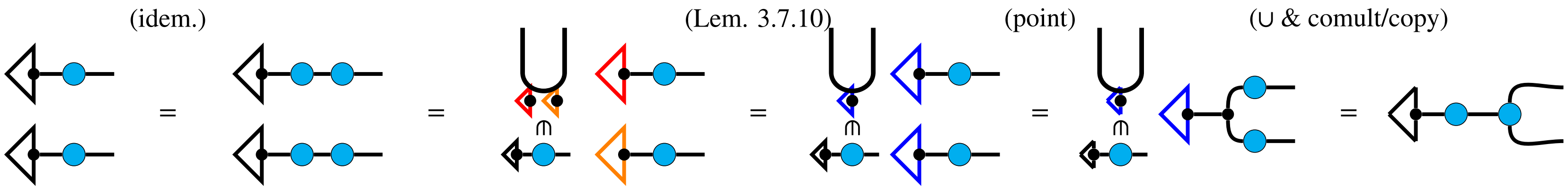


□

Proposition 3.7.11 (*e* of any point is *e*-copiable)**.**

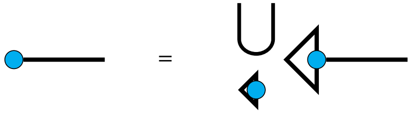


Proof.



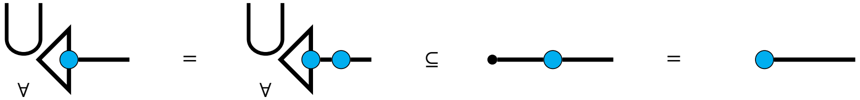
□

Proposition 3.7.12 (The unit is the union of all *e*-copiables)**.**



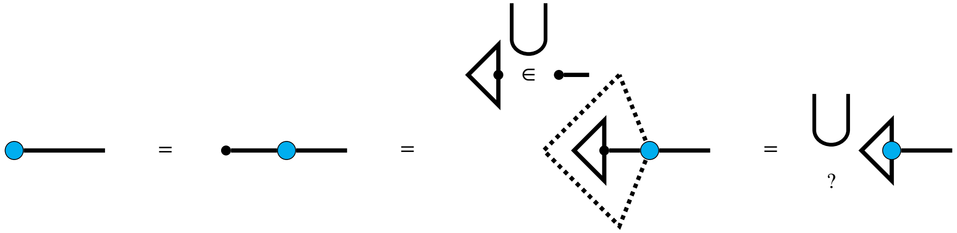
Proof.

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



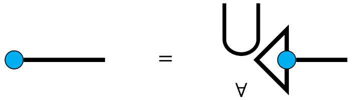
(Lem. 3.7.18) (evr.) (unit/evr.)

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



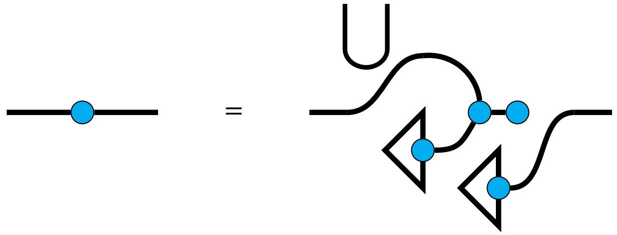
(unit/evr.) (Prop. 3.7.11)

So the containment must be an equality.

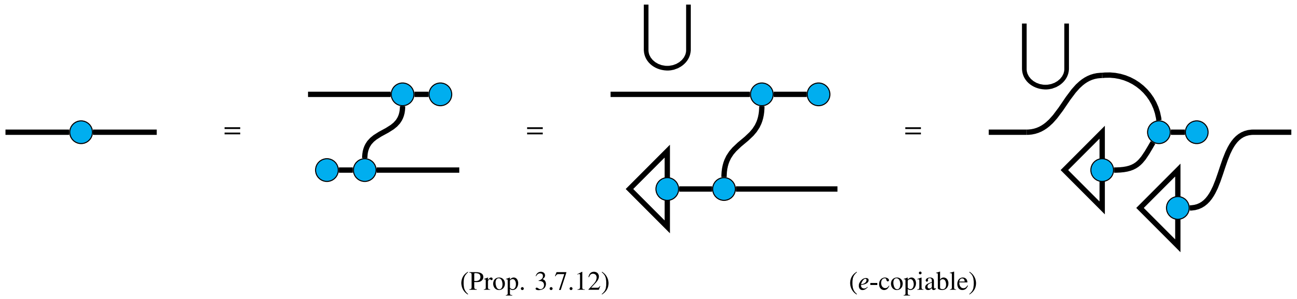


□

Proposition 3.7.13 (*e*-copiable decomposition of *e*)**.**

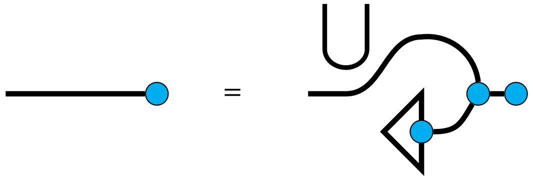


Proof.

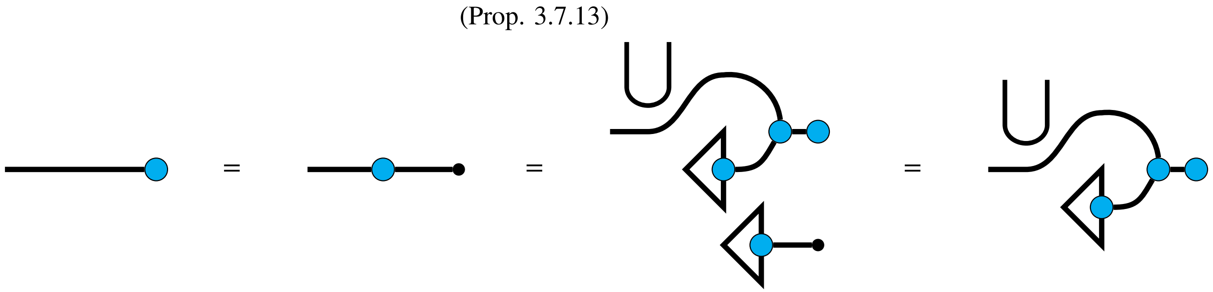


□

Proposition 3.7.14 (*e*-copiable decomposition of counit)**.**



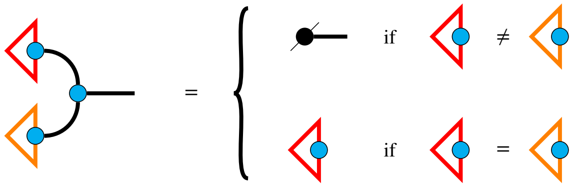
Proof.



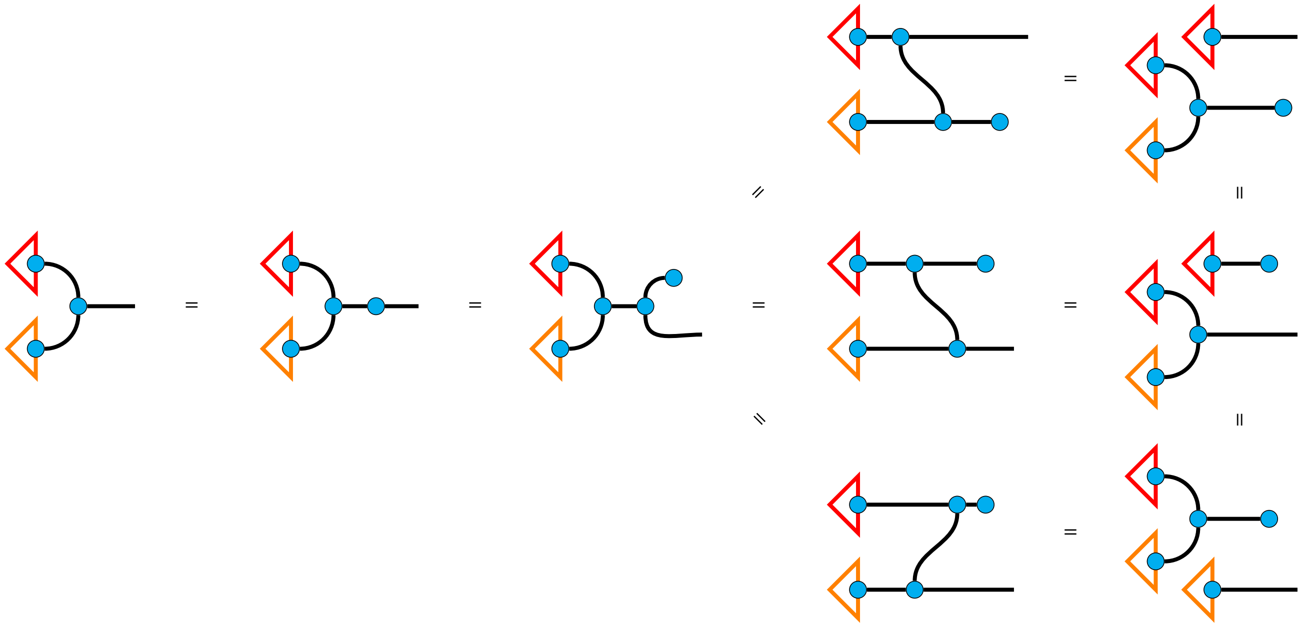
□

The *e*-copiable states really do behave like an orthonormal basis, as the following Lemmas show.

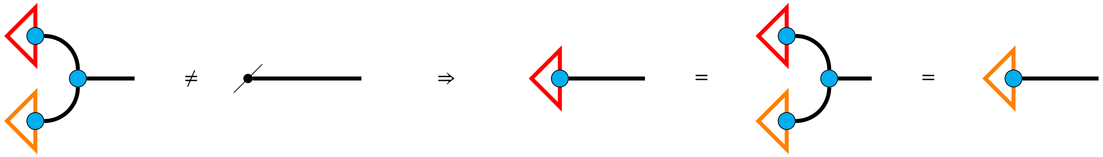
Lemma 3.7.15 (*e*-copiables are orthogonal under multiplication)**.**



Proof.

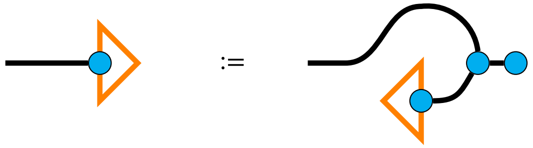


So:

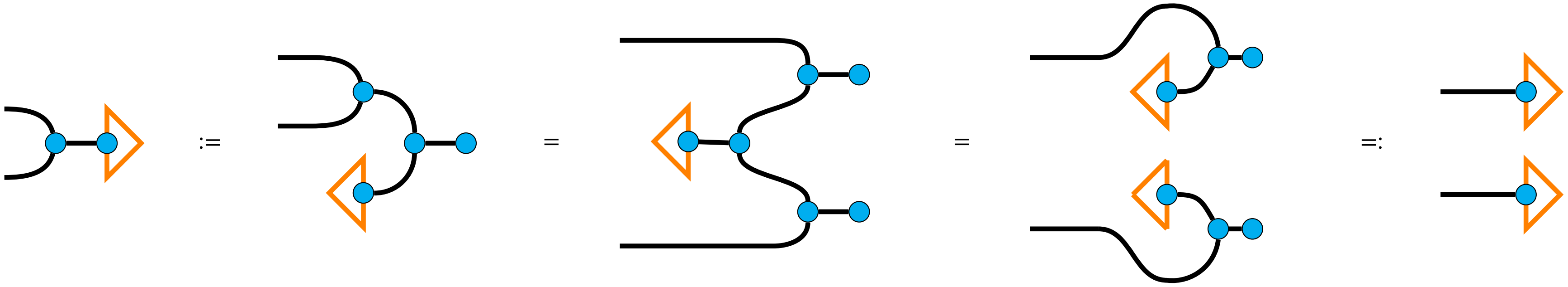


□

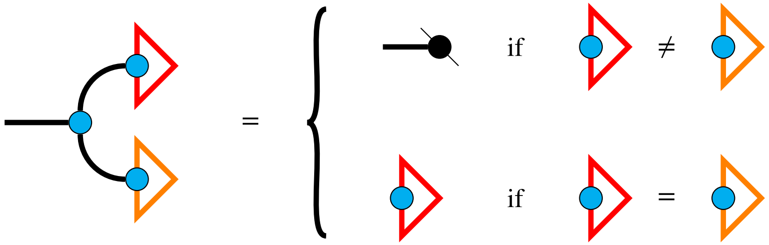
Convention 3.7.16 (Shorthand for the open set associated with an e-copiable)**.** We introduce the following diagrammatic shorthand.



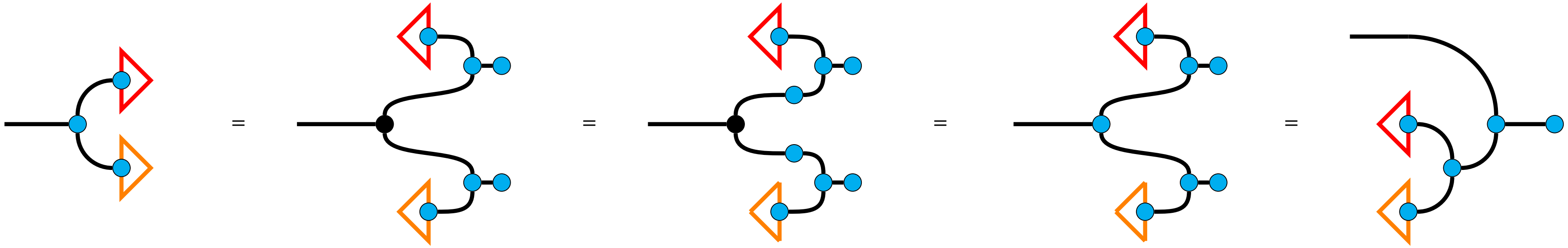
Including the coloured dot is justified, because these open sets are co-copiable with respect to the multiplication of the sticky spider.



Lemma 3.7.17 (Co-match)**.**

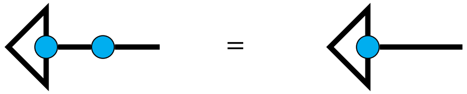


Proof.

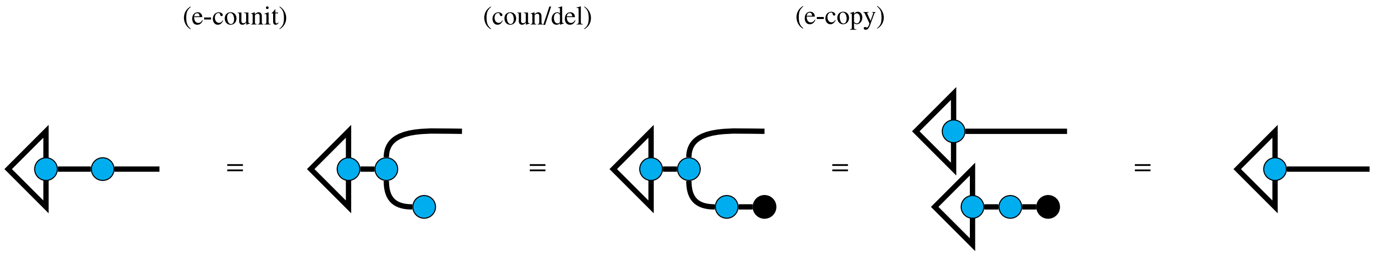


The claim then follows by applying Lemma 3.7.15 to the final diagram. □

Lemma 3.7.18 (e-copiables are e-fixpoints)**.**

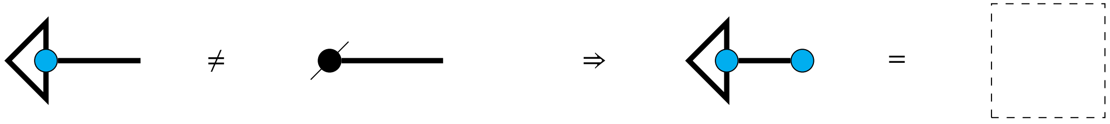


Proof.

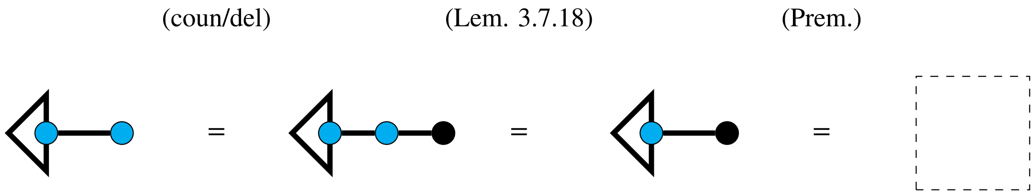


Observe that the final equation of the proof also holds when the initial e-copiable is the empty set. □

Lemma 3.7.19 (*e*-copiables are normal)**.**

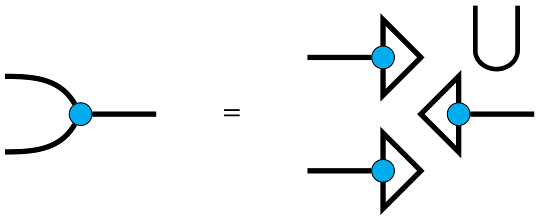


Proof.

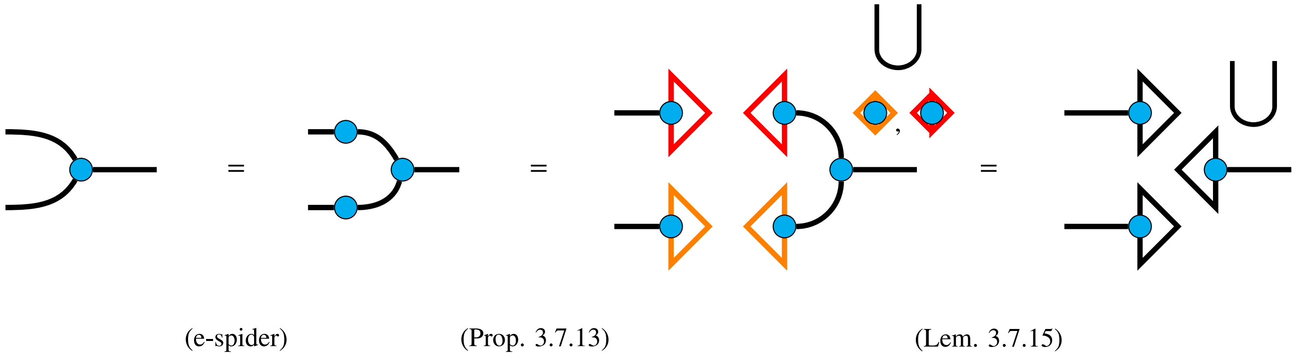


□

Proposition 3.7.20 (*e*-copiable decomposition of multiplication)**.**

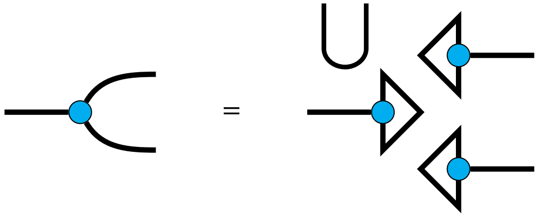


Proof.

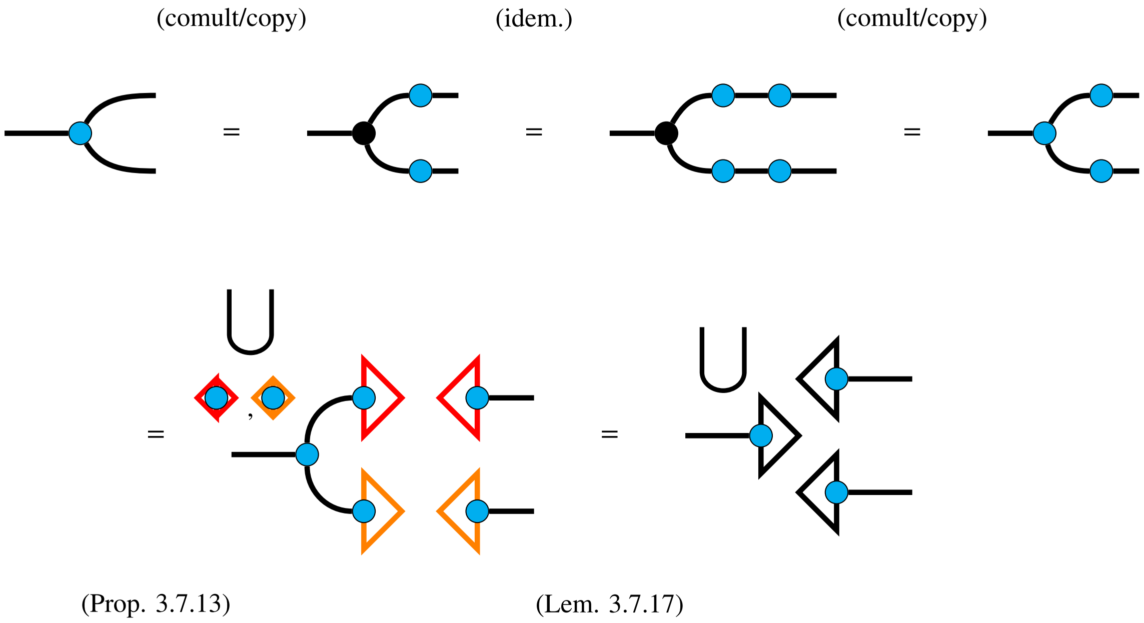


□

Proposition 3.7.21 (*e*-copiable decomposition of comultiplication)**.**



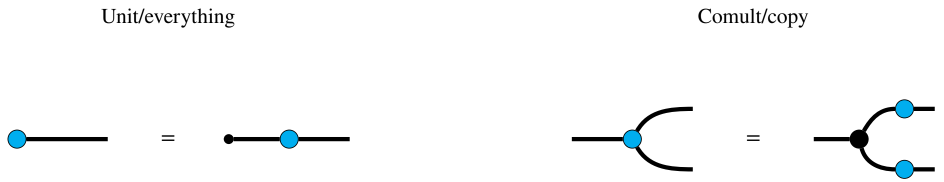
Proof.



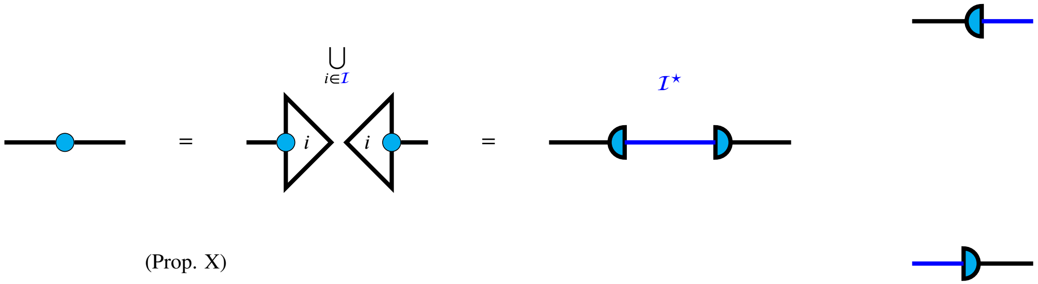
□

NOW WE CAN PROVE THEOREM 3.7.8.

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



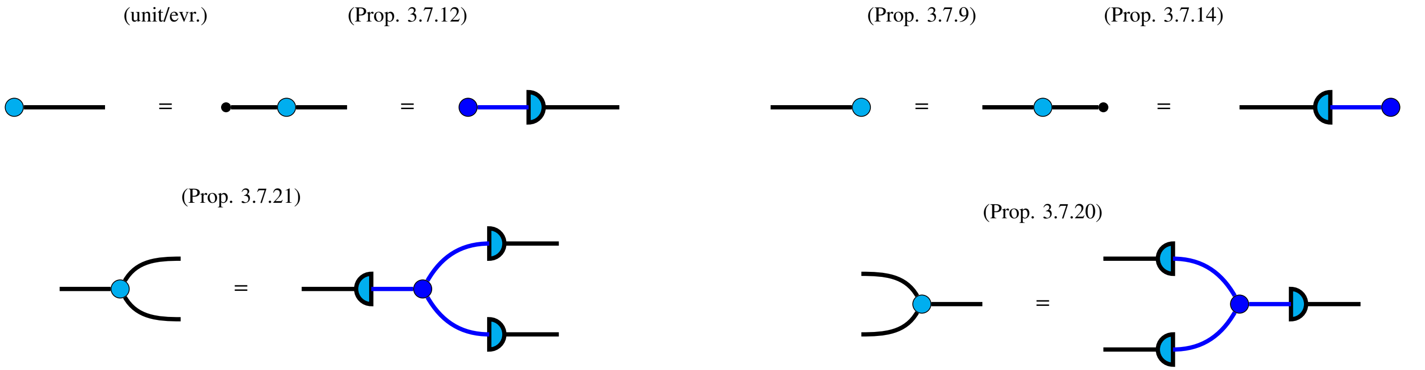
The crucial observation is that the *e*-copiable decomposition of the idempotent given by Proposition 3.7.13 is equivalent to a split idempotent though the set of *e*-copiables equipped with discrete topology.



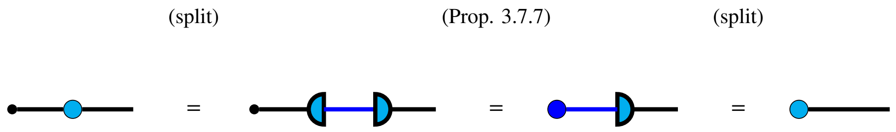
$$:= \{(x, i) \mid i \in \mathcal{I} \, , \, x \in |i >\}$$

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

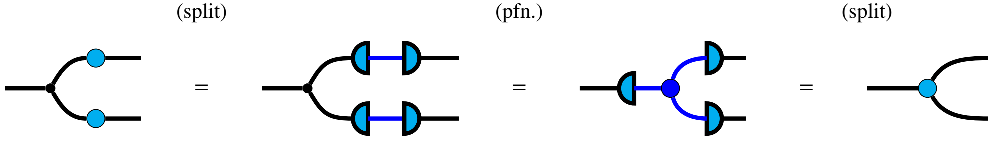
By copiable basis Proposition 3.7.12 and the decompositions Propositions 3.7.14, 3.7.20, 3.7.21, we obtain the only-if direction.



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



For the counit/delete relation, we observe that for any split idempotent, the retract must be a partial function. To see this, suppose the split idempotent $e = r; s$ is on (X, τ) and the discrete topology is $Y^\star$. Seeking contradiction, if the retract is not a partial function, then there is some point $x \in X$ such that $x \in e(x)$, and the image $I := r(x) \subseteq Y$ contains more than one point, which we denote and discriminate $a, b \in r(x) \subseteq Y$ and $a \neq b$. Because the composite $s; r = 1_Y$ of the section and retract must recover the identity on $Y^\star$, the section s must be total – i.e. the image $s(X) = Y$. So $x \in s(a) \cap s(b)$. Now we have that $(a, x), (b, x) \in s$, and $(x, a), (x, b) \in r$, therefore $(a, b), (b, a) \in s; r$, which by $a \neq b$ contradicts that $s; r$ is the identity relation 1_Y.



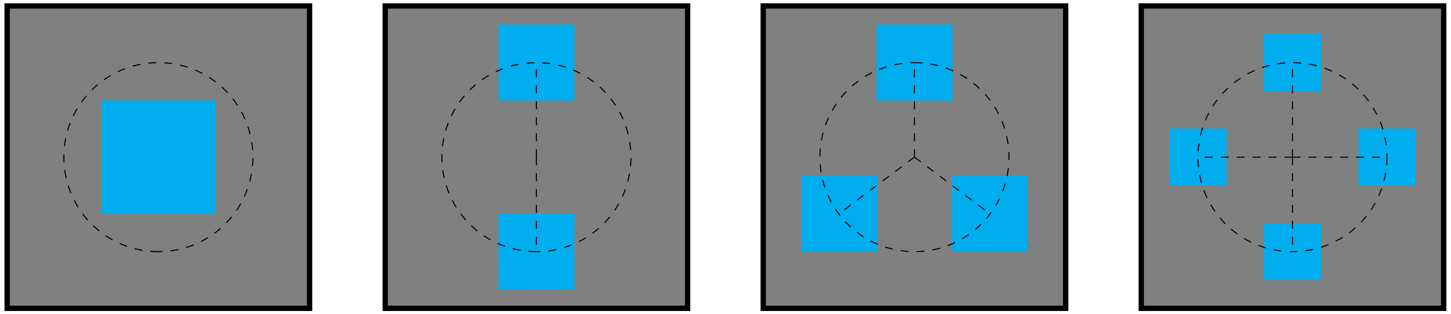
□

3.8 Modelling Lassos

Recall that Lassos – a graphical gadget that can encode arbitrary morphisms into a single wire – can be interpreted in a monoidal computer. Recall that monoidal computers require a universal object Ξ. Here we show how in **TopRel**, by taking $\Xi :=$ ▦ the open unit square, we have a monoidal computer in **Rel** restricted to countable sets and the relations between them. We will make use of sticky spiders. We have to show that; ▦ has a sticky-spider corresponding to every countable set; how there is a suitable notion of sticky-spider morphism to establish a correspondence with relations; what the continuous relations are on ▦ that mimick various compositions of relations.

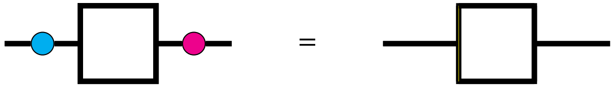
Proposition 3.8.1 ($(0,1) \times (0,1)$ splits through any countable set X)**.** For any countable set X, the open unit square ▦ has a sticky spider that splits through $X^\star$.

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



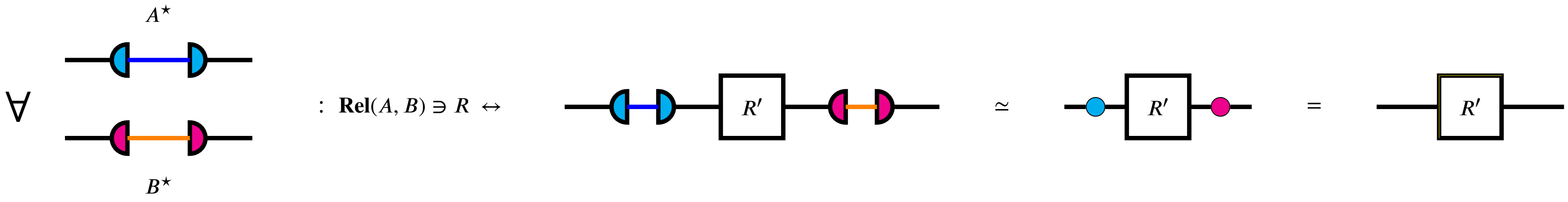
□

Definition 3.8.2 (Morphism of sticky spiders)**.** A morphism between sticky spiders is any morphism that satisfies the following equation.



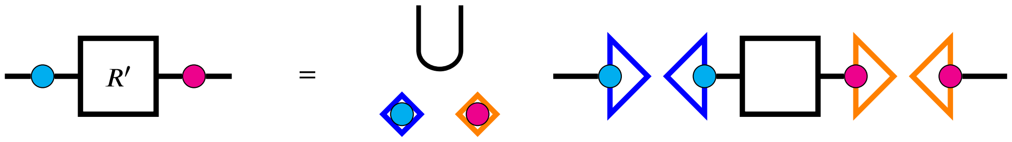
Proposition 3.8.3 (Morphisms of sticky spiders encode relations)**.** For arbitrary split idempotents through $A^\star$ and

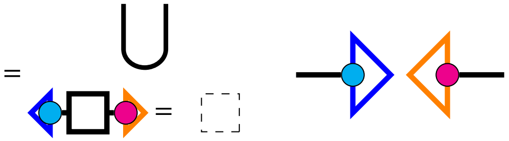
$B^\star$, the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \to B$.



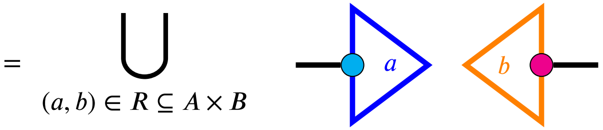
Proof.

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

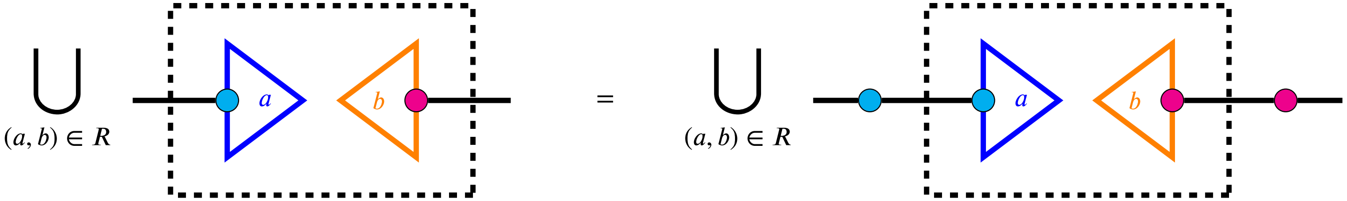




Since (co)copiables are distinct, we may uniquely reindex as:

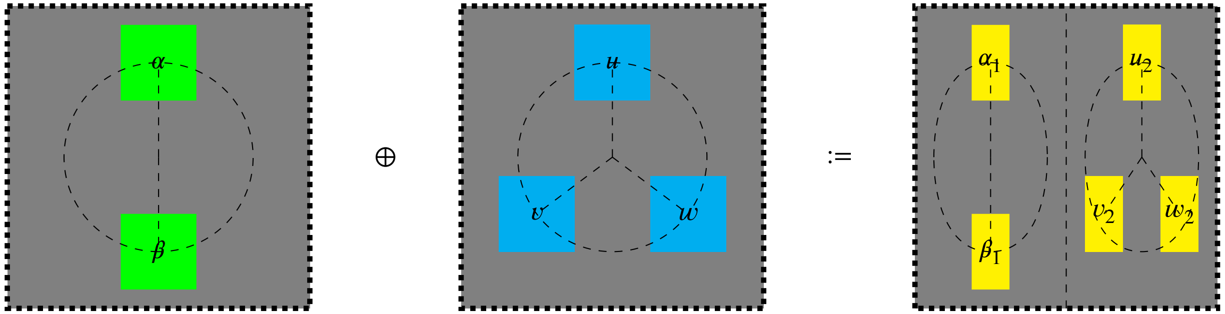


(⇒) : By idempotence of (co)copiables, every relation $R \subseteq A \times B$ corresponds to a morphism of sticky spiders.

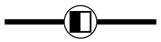


□

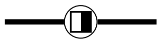
Construction 3.8.4 (Representing sets in their various guises within ▦)**.** We can represent the direct sum of two ▦-representations of sets as follows.



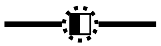
The important bit of technology is the homeomorphism that losslessly squishes the whole unit square into one half of the unit square. The decompressions are partial continuous functions, with domain restricted to the appropriate half of the unit square.



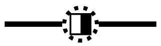
$(x, y) \mapsto (\frac{x}{2}, y)$



$(x, y) \mapsto (\frac{x+1}{2}, y)$

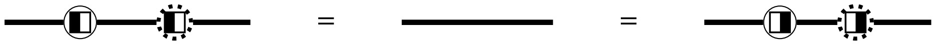


$(x, y)|_{x<\frac{1}{2}} \mapsto (2x, y)$

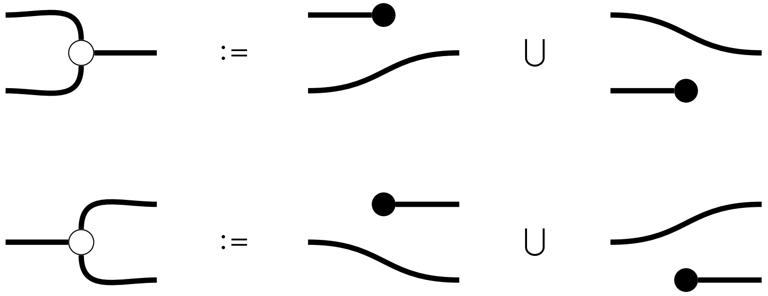


$(x, y)|_{x>\frac{1}{2}} \mapsto (2x - 1, y)$

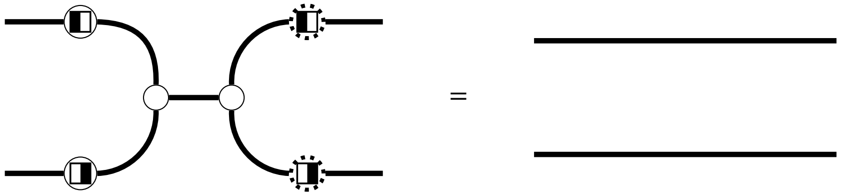
We express the ability of these relations to encode and decode the unit square in just either half by the following graphical equations.



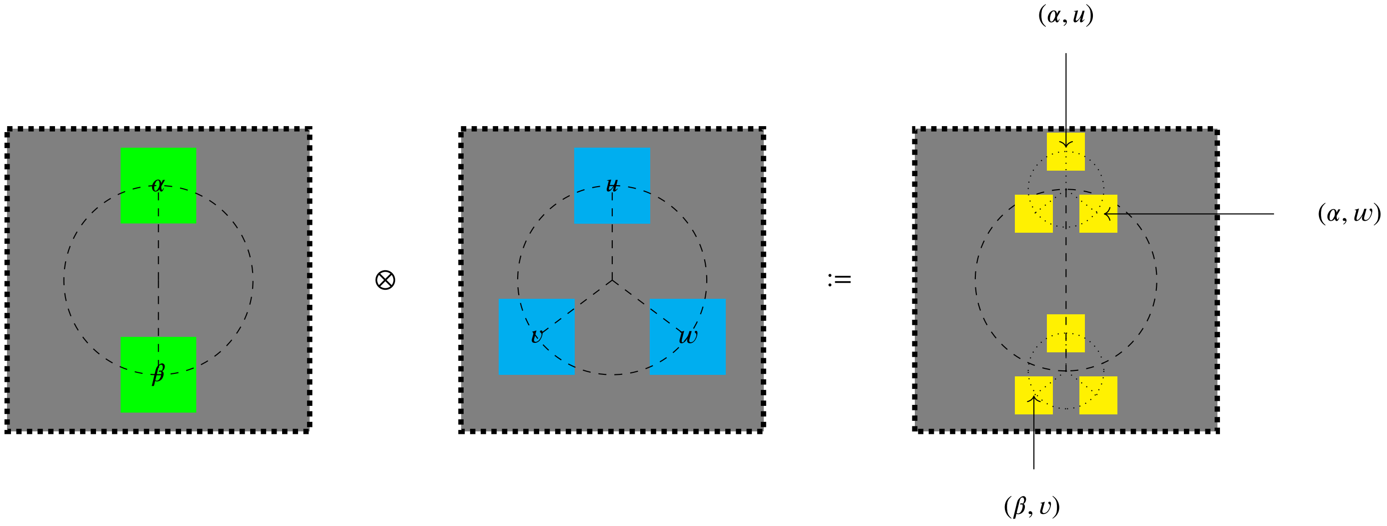
Now, to put the two halves together and to take them apart, we introduce the following two relations. In tandem with the squishing and stretching we have defined, these will behave just as the projections and injections for the direct-sum biproduct in **Rel**.



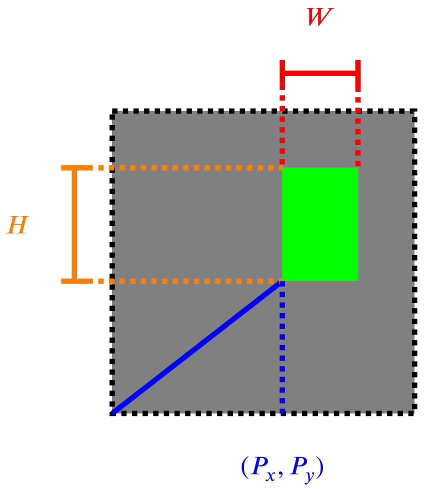
The following equation tells us that we can take any two representations in ▦, put them into a single copy of ▦, and take them out again. Banach and Tarski would approve.

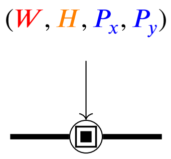


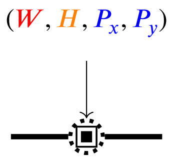
We encode the tensor product $A \otimes B$ of representations by placing copies of B in each of the open boxes of A.



The important bit of technology here is a family of homeomorphisms of ▦ parameterised by axis-aligned open boxes. We depict the parameters outside the body of the homeomorphism for clarity. The squish is on the left, the stretch on the right.



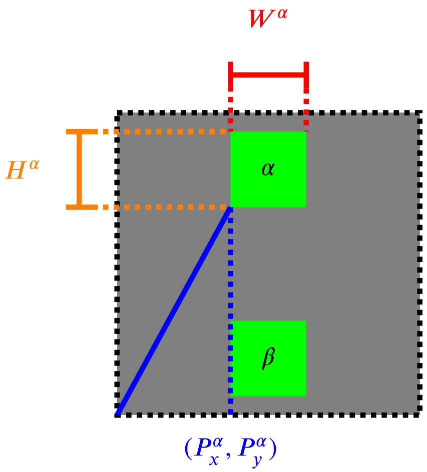




$$(x, y) \mapsto (Wx + P_x, Hy + P_y)$$

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

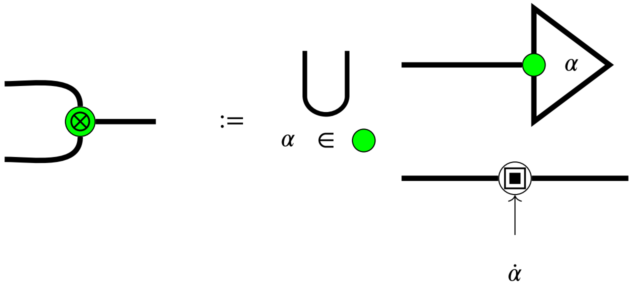
Now, for every representation of a set in ▦ by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a squish-stretch homeomorphism via the parameters of the open box, which we notate with a dot above the name of the element.

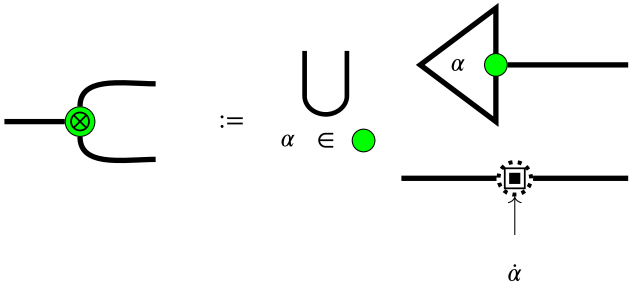


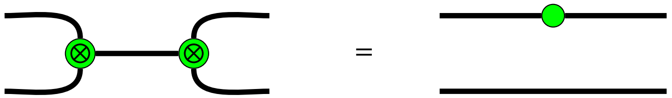
$$\dot{\alpha} = (W^\alpha, H^\alpha, P^\alpha_x, P^\alpha_y)$$

$$\dot{\beta} = (W^\beta, H^\beta, P^\beta_x, P^\beta_y)$$

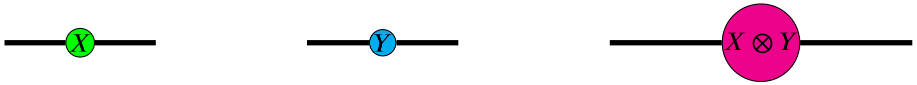
Now we can define the "tensor X on the left" relation _ → $X \otimes$ _ and its corresponding cotensor.

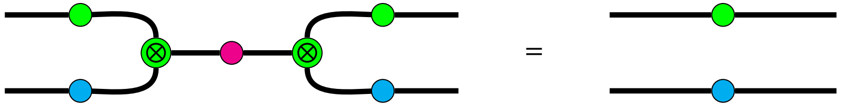


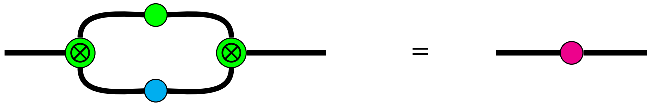




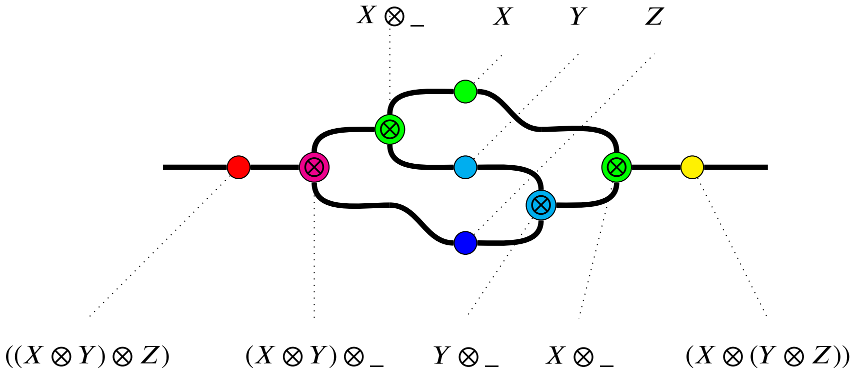
The tensor and cotensor behave as we expect from proof nets for monoidal categories.



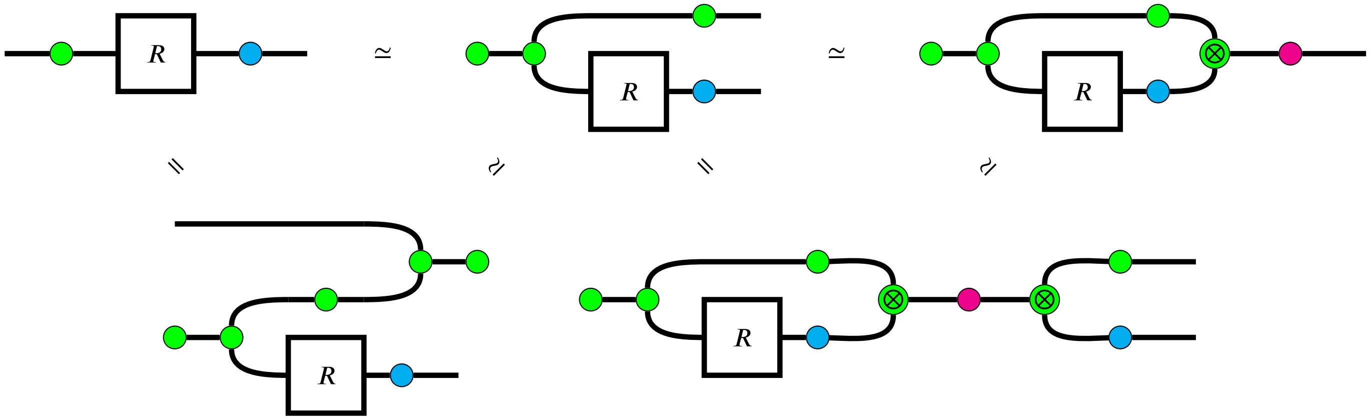




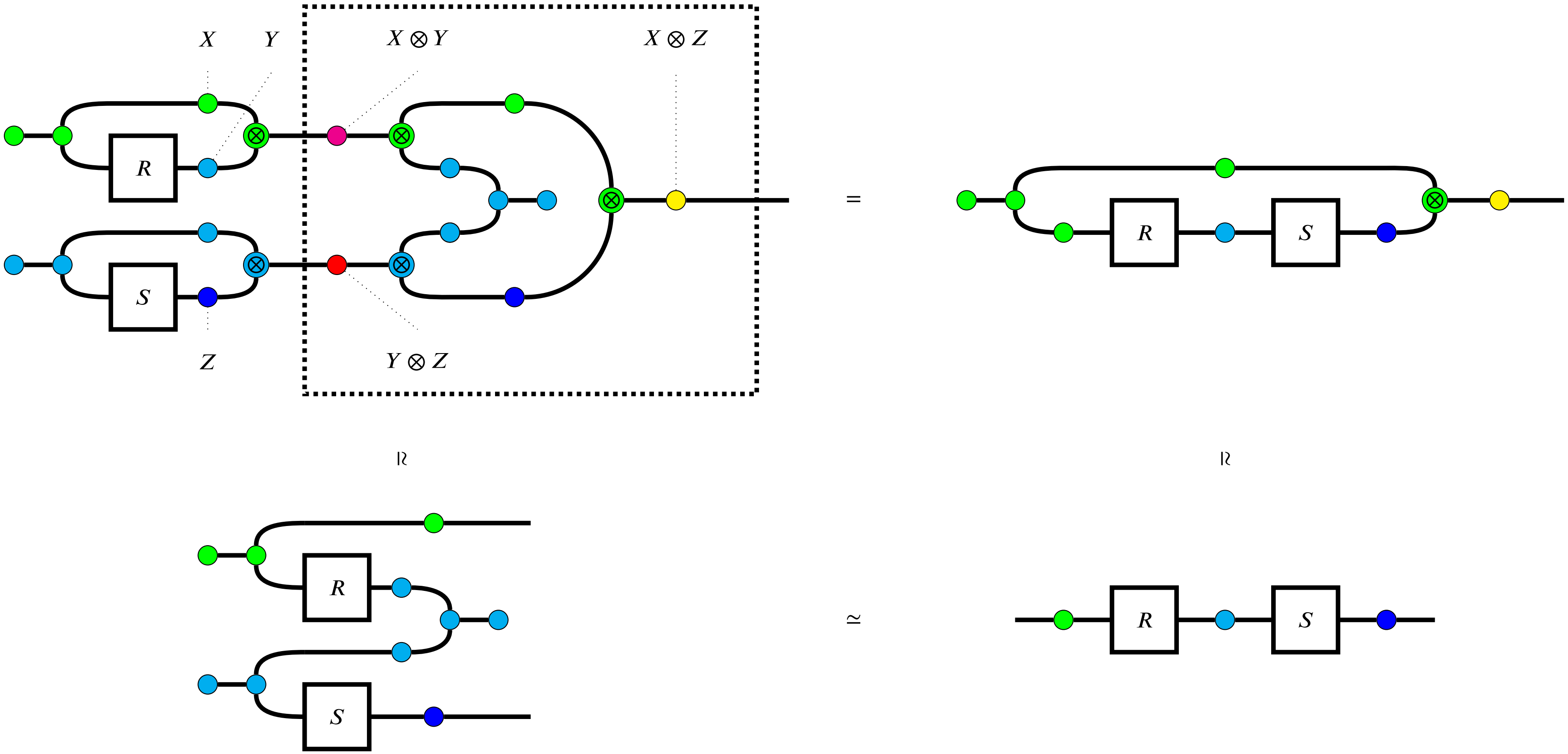
And by construction, the (co)tensors and (co)pluses interact as we expect, and they come with all the natural isomorphisms between representations we expect. For example, below we exhibit an explicit associator natural isomorphism between representations.


$X \otimes \_$
X
Y
Z
$((X \otimes Y) \otimes Z)$
$(X \otimes Y) \otimes \_$
$Y \otimes \_$
$X \otimes \_$
$(X \otimes (Y \otimes Z))$

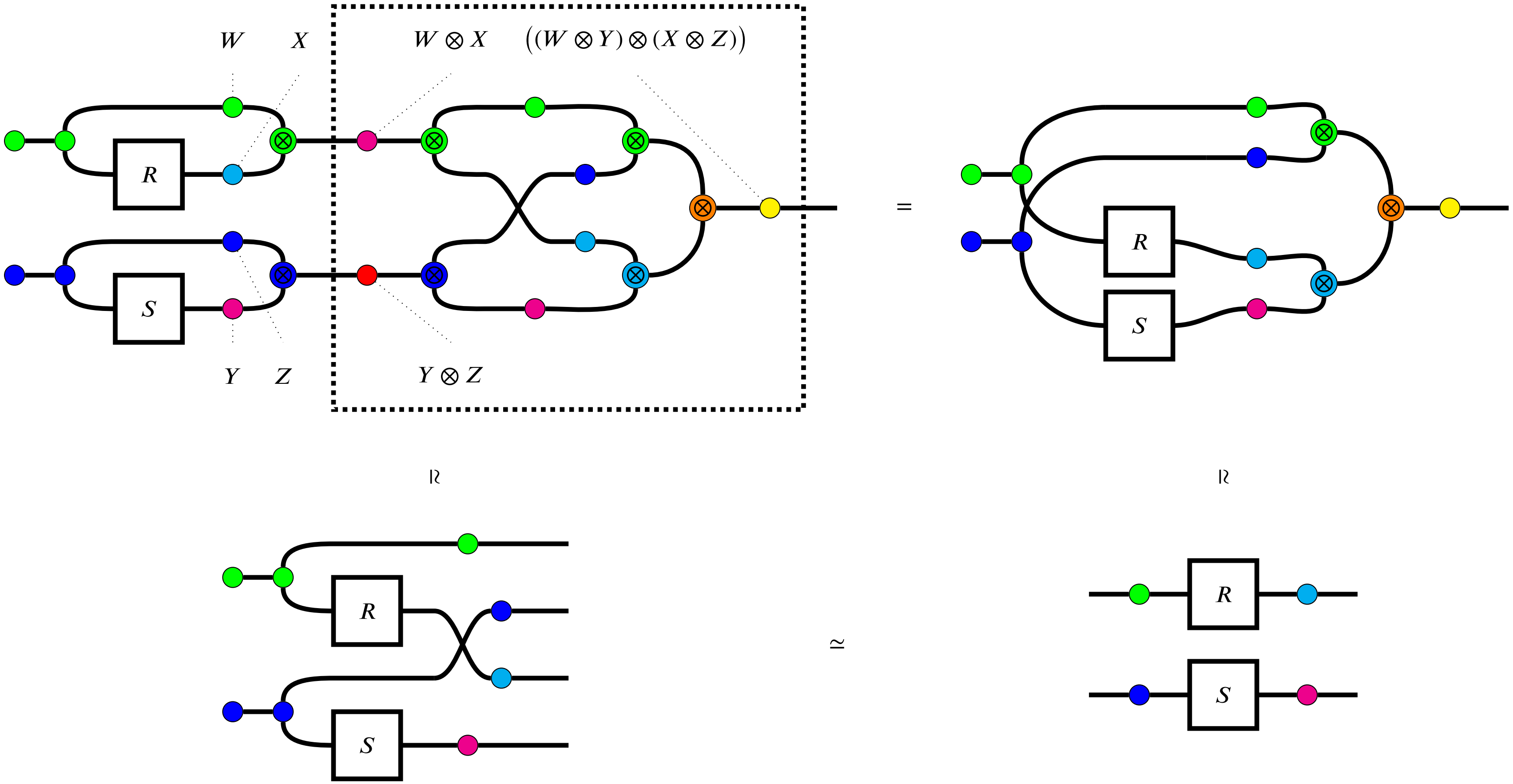
Construction 3.8.5 (Representing relations between sets and their composition within ▦)**.** With all the above, we can establish a special kind of process-state duality; relations as processes are isomorphic to states of ▦, up to the representation scheme we have chosen.



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



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

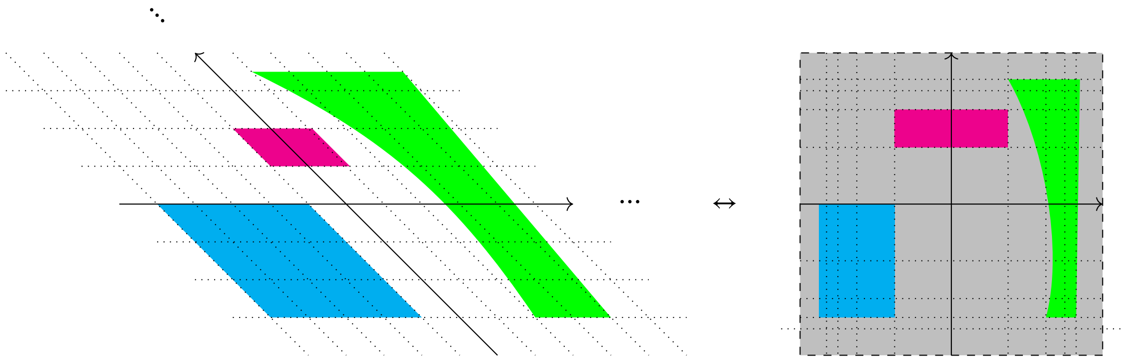


3.9 Topological concepts in flatland via ***TopRel***

3.9.1 Shapes and places

Definition 3.9.1 (Shapes, cores, halos)**.**
Proposition 3.9.2 (Core exclusion: Cores cannot overlap)**.**
Proposition 3.9.3 (Core-halo exclusion: Each core only overlaps with its corresponding halo)**.**
Proposition 3.9.4 (Halo non-exclusion: Cores cannot overlap)**.**

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



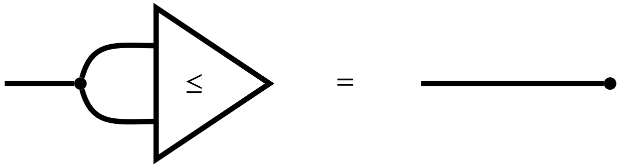
3.9.2 The unit interval

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

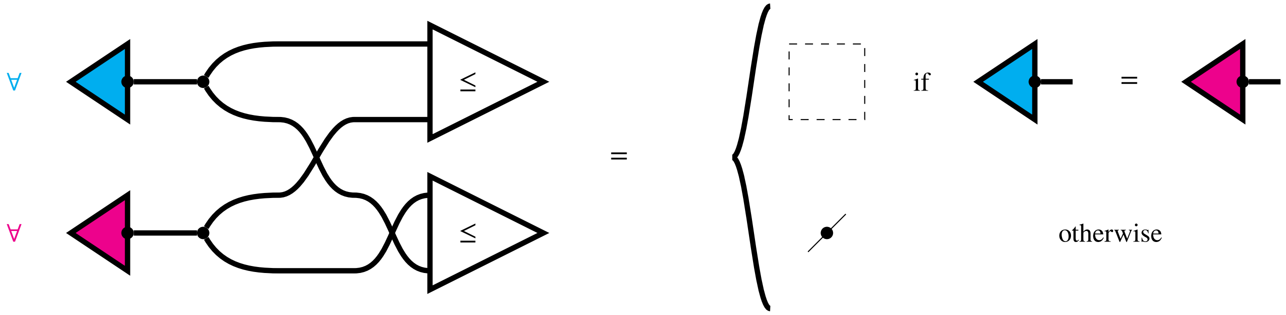
Theorem 3.9.5 (Friedman)**.** Let $\big((X, \tau), <\big)$ be a topological space with a total order. If there exists a continuous map $f : X \times X \to X$ such that $\forall a, b_{\in X} : a < f(a, b) < b$, then X is homeomorphic to $\boldsymbol{R}$.

We can define all of these pieces using diagrammatic equations. LESS THAN We define $<$ as an open set on $X \times X$ that obeys the usual axiomatic rules:

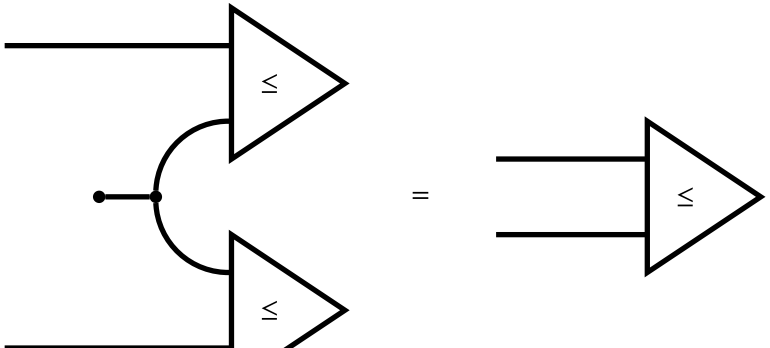
Reflexive



Antisymmetric



Transitive



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

Definition 3.9.6 (Locally indiscrete topology)**.** (X, τ) is *locally indiscrete* when every open set is also closed.

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

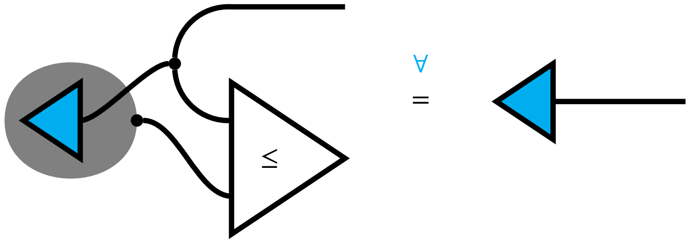
$$placeholder$$

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

Definition 3.9.7 (Partial endocombinator)**.** In a category $\mathcal{C}$, a *partial endocombinator* on a homset $(\mathcal{C})(A, B)$ is a function $(\mathcal{C})(A, B) \to (\mathcal{C})(A, B)$

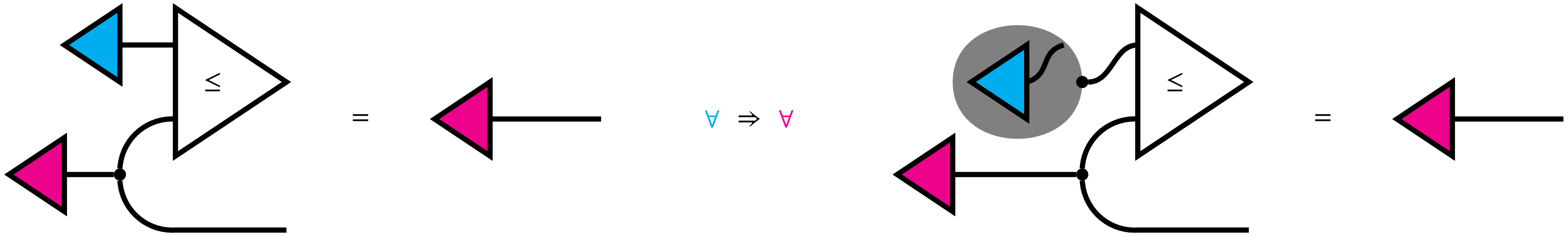
Using this technology, we can define:

Upper Bound

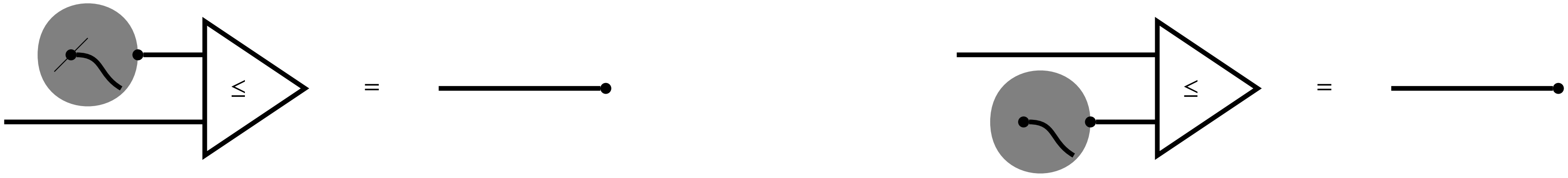


And we can add in further equations governing the upper bound endocombinator to turn it into a supremum, where the lower endpoint is obtained as the supremum of the empty set, and the upper endpoint is the supremum of the whole set.

Supremum



Now we can define endpoints purely graphically:



Lower endpoint

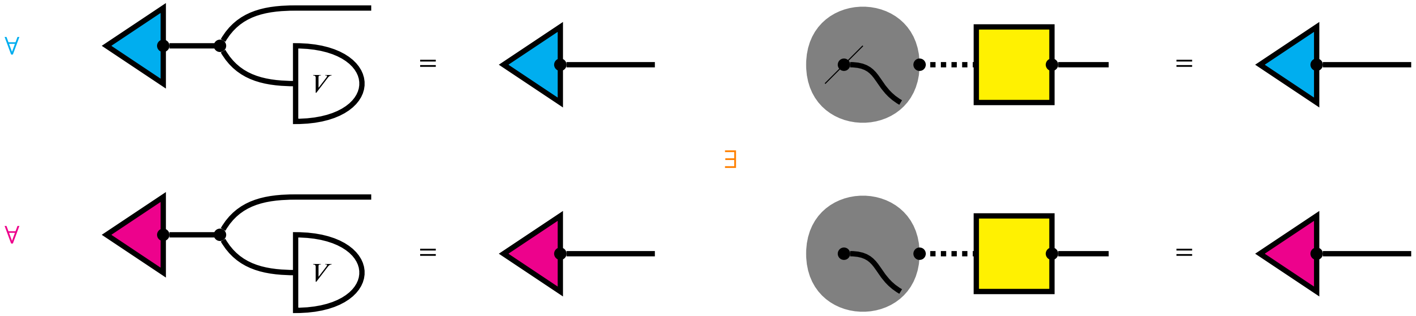
Upper endpoint

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

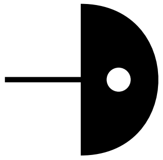
Simply connected spaces

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

V is *simply connected* when:



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



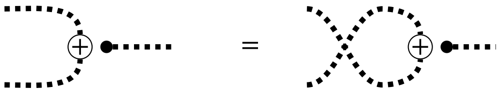
3.9.3 Displacing shapes

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

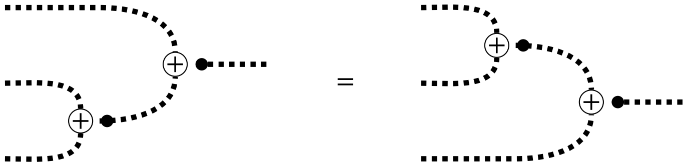
Addition

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

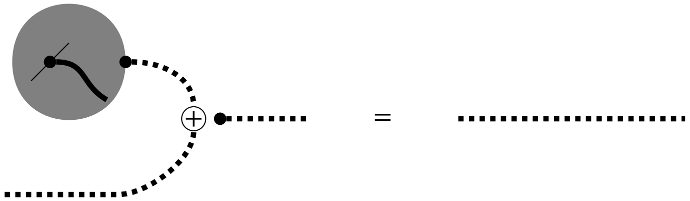
Symmetric



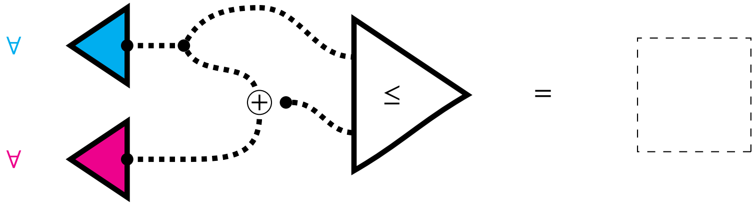
Associative



Unital (with lower endpoint)



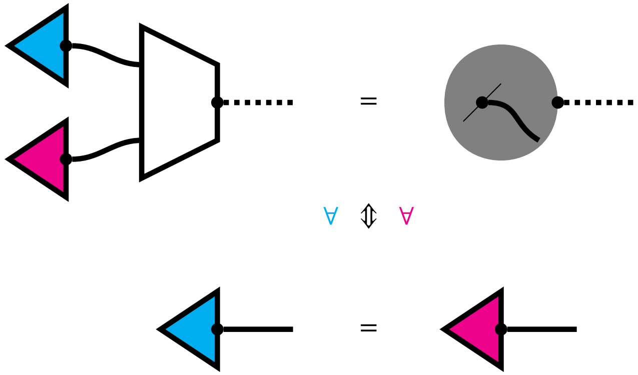
Monotone



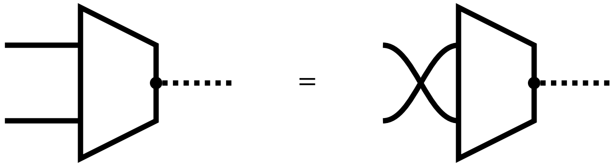
Metrics

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

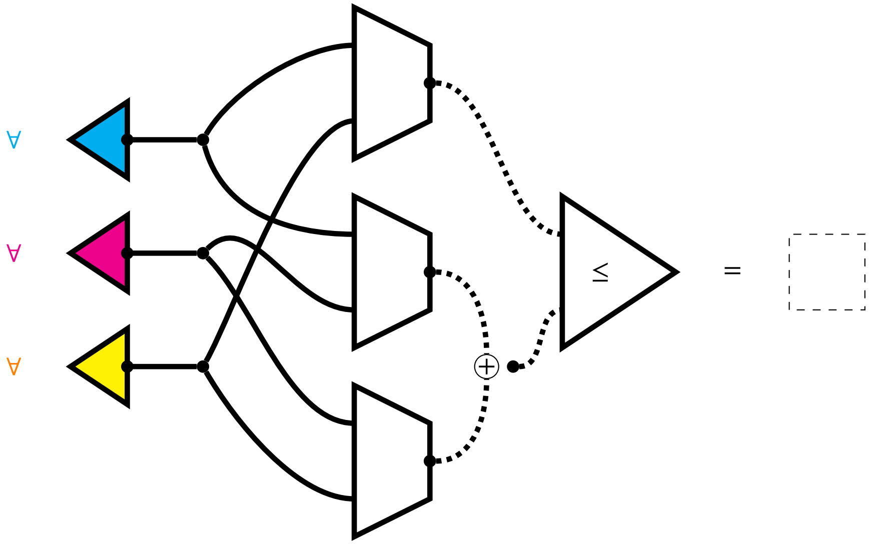
$$d(x, y) = 0 \iff x = y$$



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



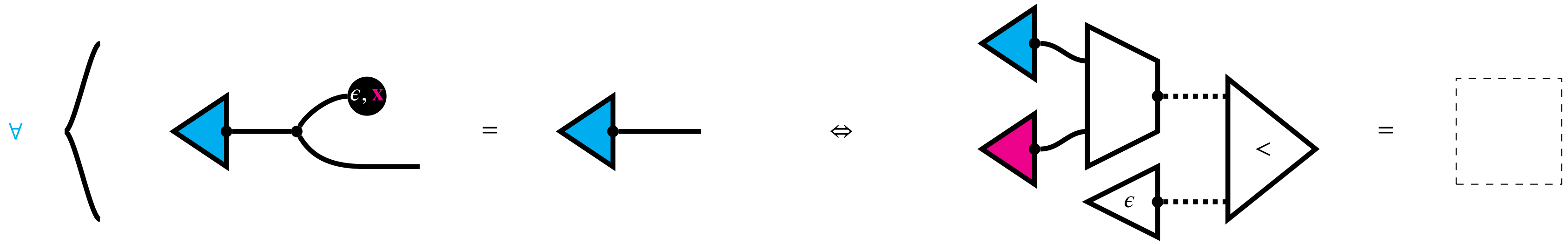
$$d(x, z) \leq d(x, y) + d(y, z)$$



Open balls

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

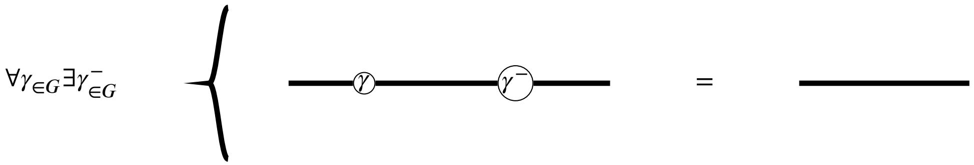
Open ball of radius ϵ at a point **x**



Topological Group

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

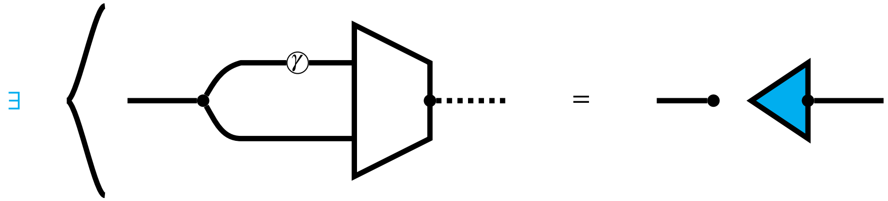
A topological group G



Isometry

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

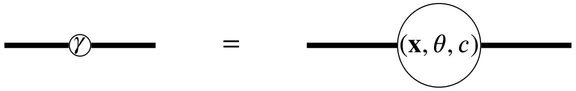
γ is an *isometry*



Rigid displacements

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

Isometries of $\mathbf{R}^2$



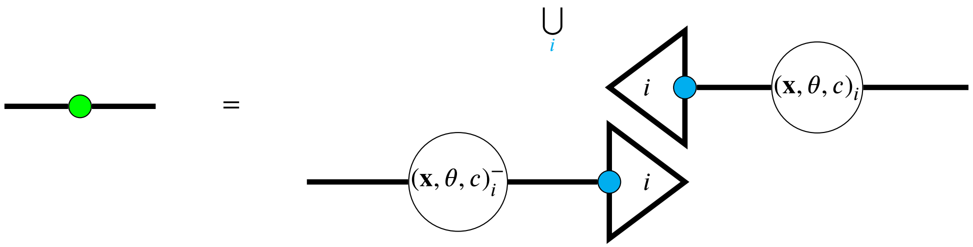
$$\mathbf{x} \in \mathbf{R}^2$$

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

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

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

Rigid displacement



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

3.9.4 Moving shapes

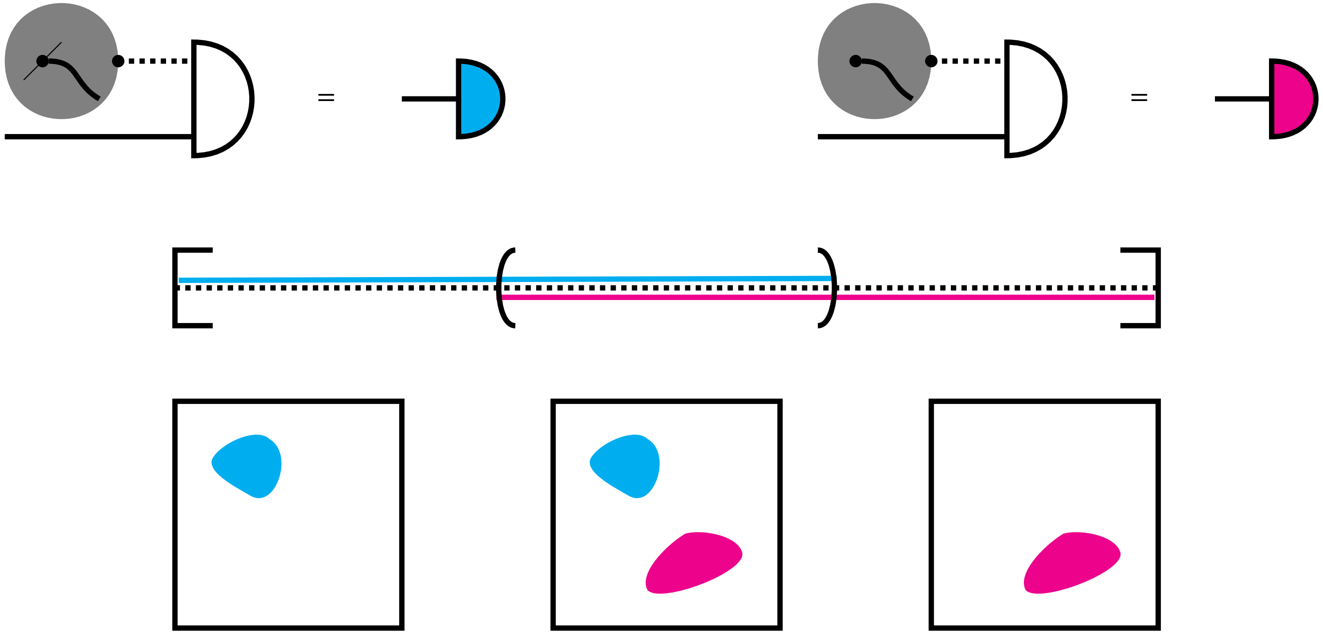
If we want continuous transformations in the plane from the configuration of shapes in one spider to end at the configuration of shapes in another, we ought to define an analogue of *homotopy*: the continuous deformation of one map to another. However, we will have to massage the definition a little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

Homotopy in **TopRel**

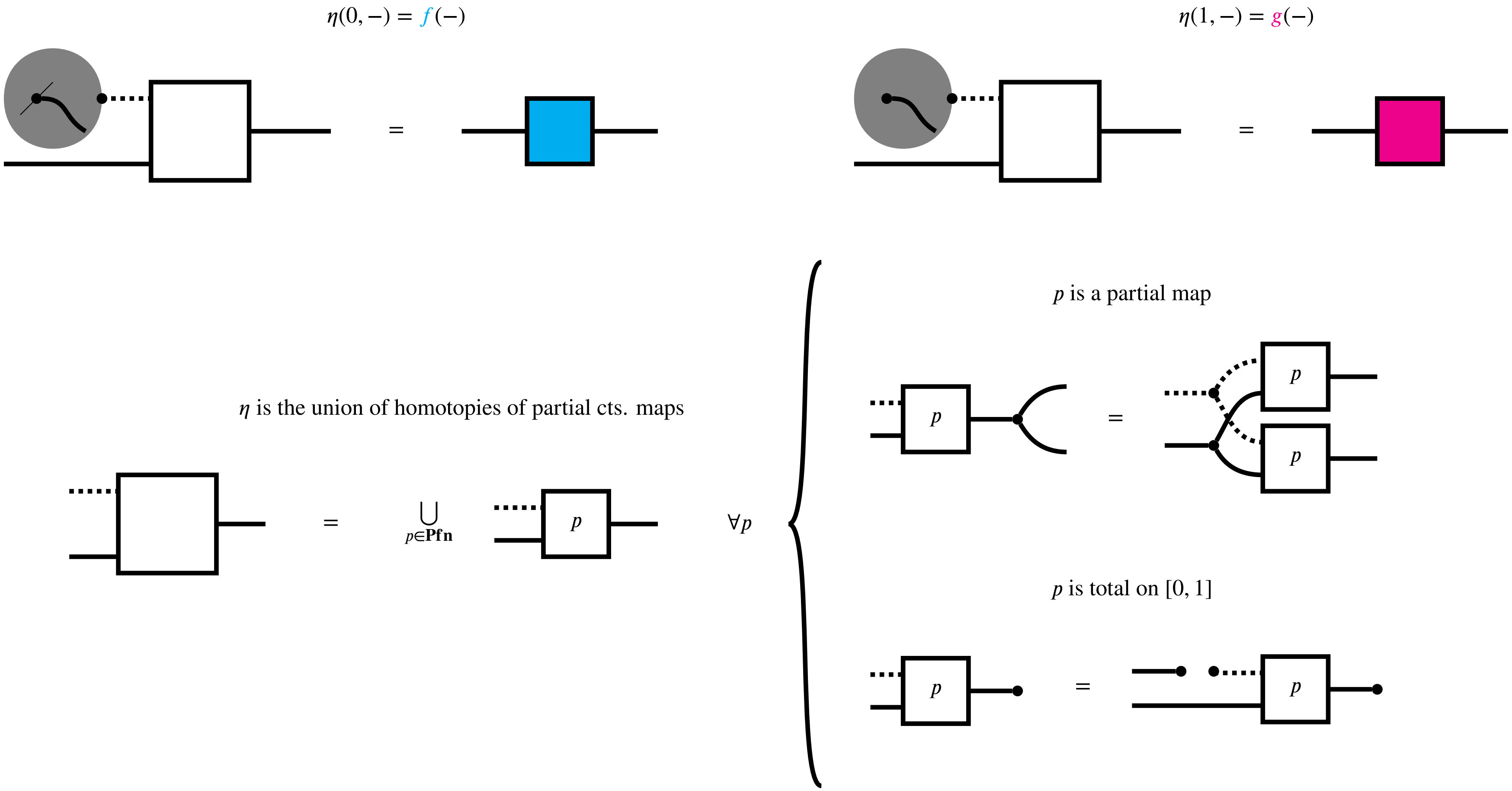
Usually, when we are restricted to speaking of topological spaces and continuous functions, a homotopy is defined:

Definition 3.9.8 (Homotopy in **Top**)**.** where f and g are continuous maps $A \to B$, a *homotopy* $\eta : f \Rightarrow g$ is a continuous function $\eta : [0,1] \times A \to B$ such that $\eta(0,-) = f(-)$ and $\eta(1,-) = g(1,-)$.

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



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



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

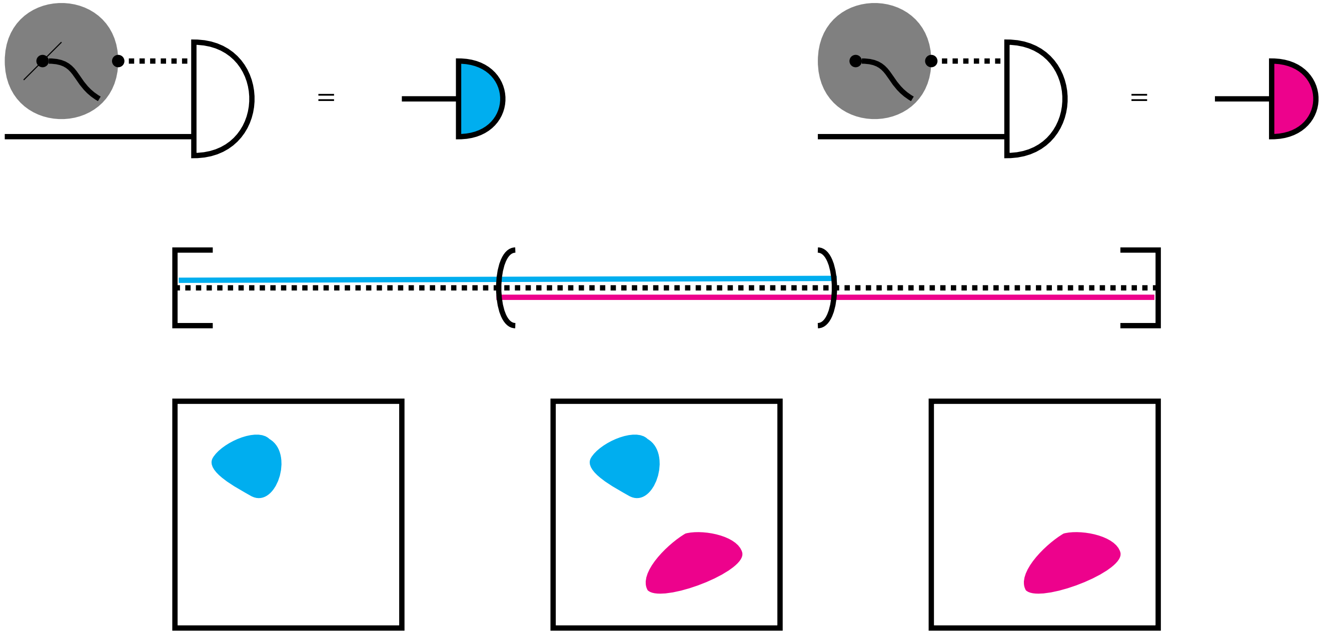
$\eta(1,-) = g(-)$

η is the union of homotopies of partial cts. maps

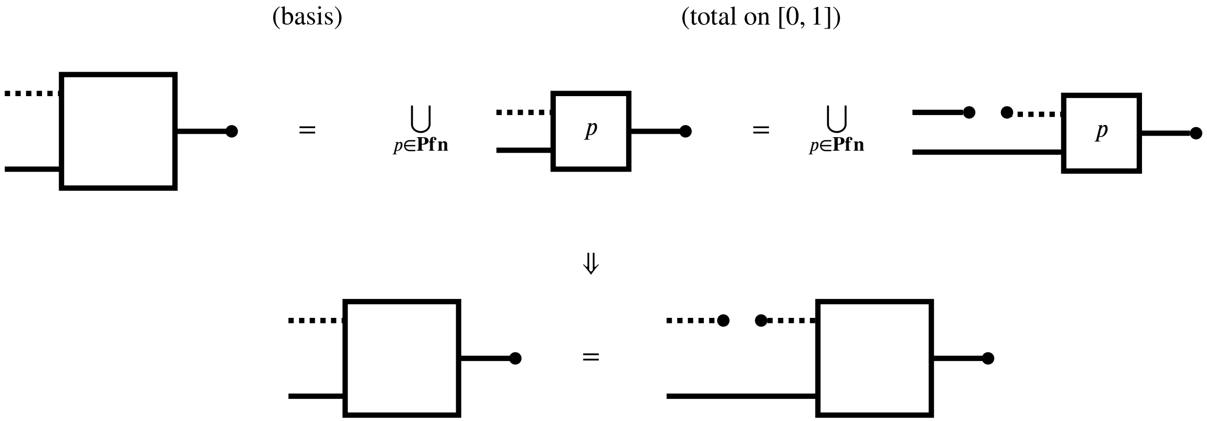
p is a partial map

p is total on $[0,1]$

Observe that the second condition asking for decomposition in terms of partial comes for free by Proposition 3.2.20; the constraint of the definition is provided by the first condition, which is a stronger condition than just asking that the original continuous relation be total on I:



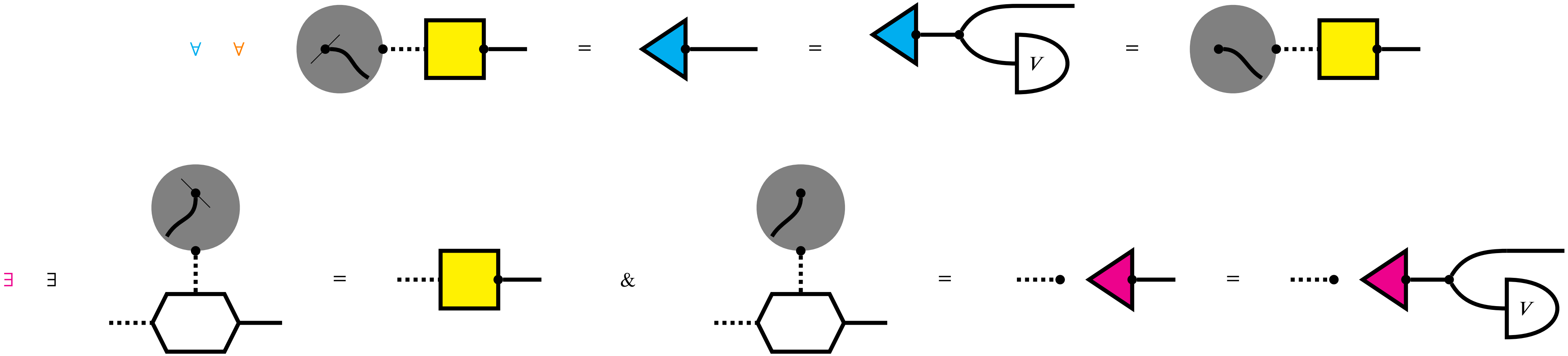
This definition is "natural" in light of Proposition 3.2.20, that the partial continuous functions $A \to B$ form a basis for $\mathbf{TopRel}(A, B)$: we are just asking that homotopies between partial continuous functions – which can be viewed as regular homotopies with domain restricted to the subspace topology induced by an open set – form a basis for homotopies between continuous relations.



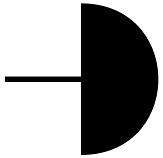
Contractible spaces

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

V is *contractible* when:



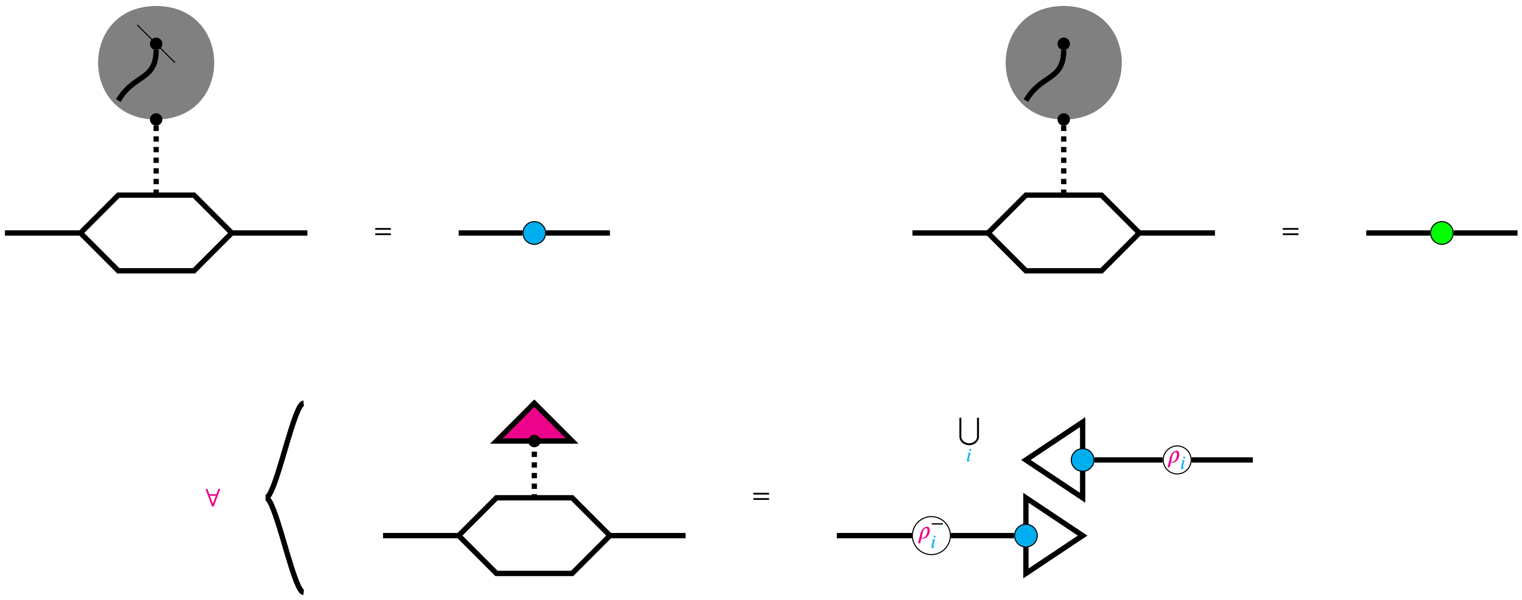
Contractible open sets are worth their own notation too; a solid black effect, this time with no hole.



3.9.5 *Rigid motion*

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

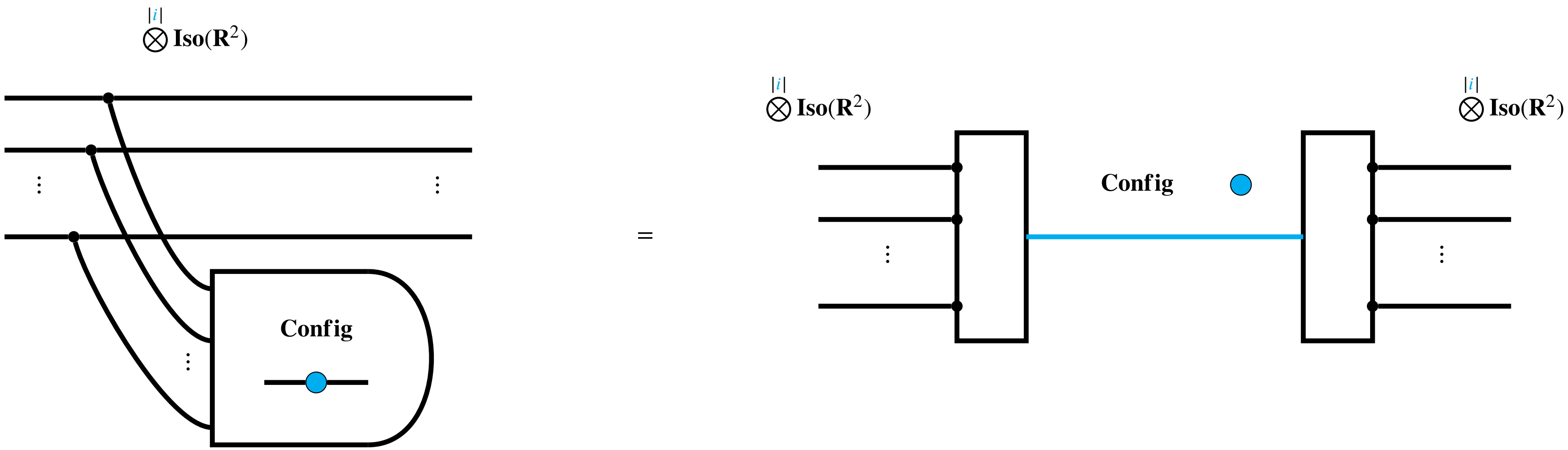
Rigid motion



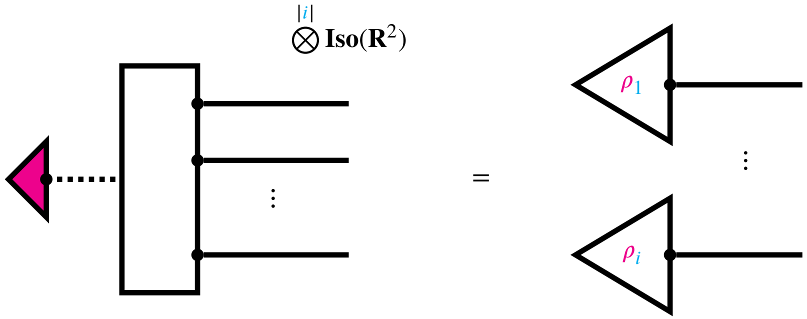
CONFIGURATION SPACES

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

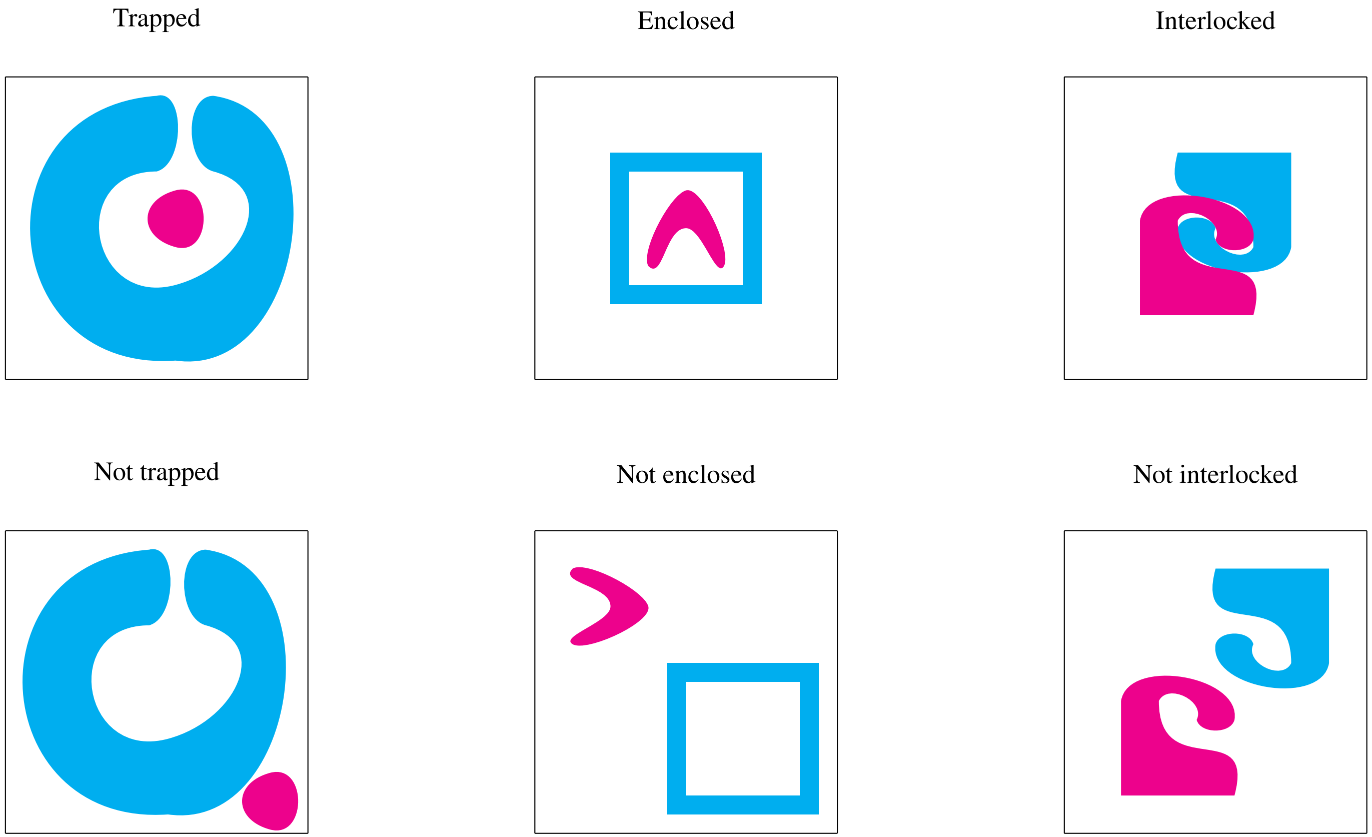
Configuration space of a sticky spider



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



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



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

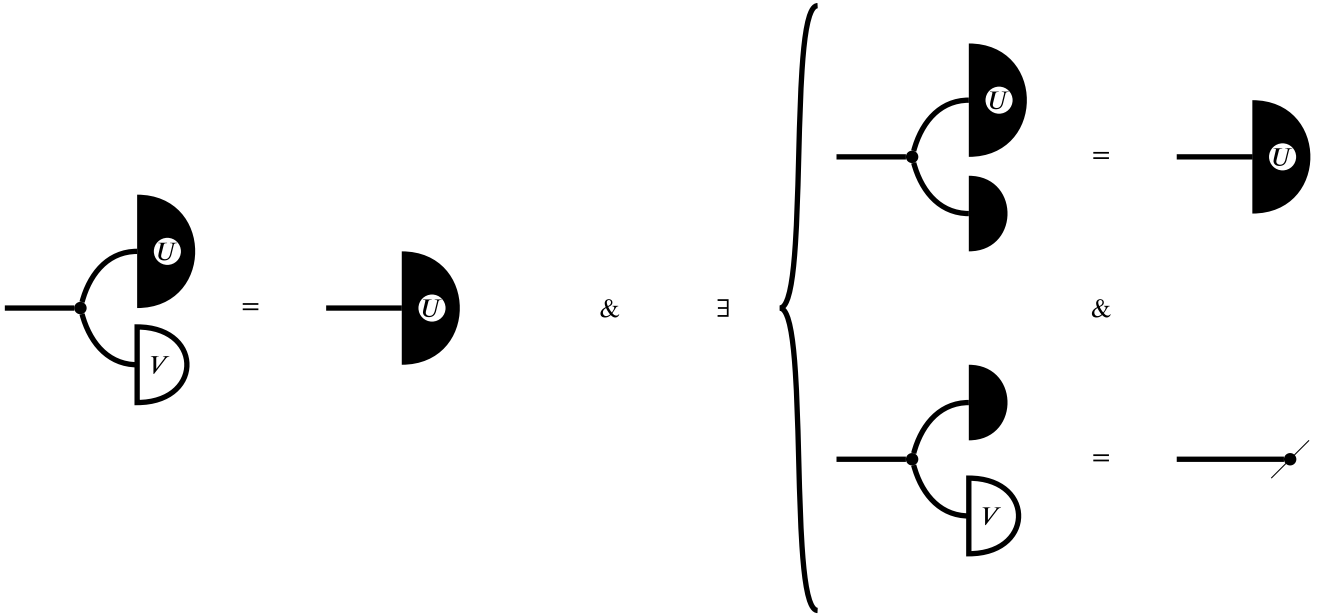
3.9.6 Modelling linguistic topological concepts

By "linguistic", I mean to refer to the kinds of concepts we use in everyday language. These are concepts that even young children have an intuitive grasp of [], but their formal definitions are difficult to pin down. One such relation modelled here – touching – is in fact a *semantic prime* []: a word that is present in essentially all natural languages that is conceptually primitive, in the sense that it resists definition in simpler terms. It is among the ranks of concepts like *wanting* or *living*, words that are understood by the experience of being human, rather than by school. As such, I make no claim that these definitions are "correct" or "canonical", just that they are good enough to build upon moving forward.

PARTHOOD

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

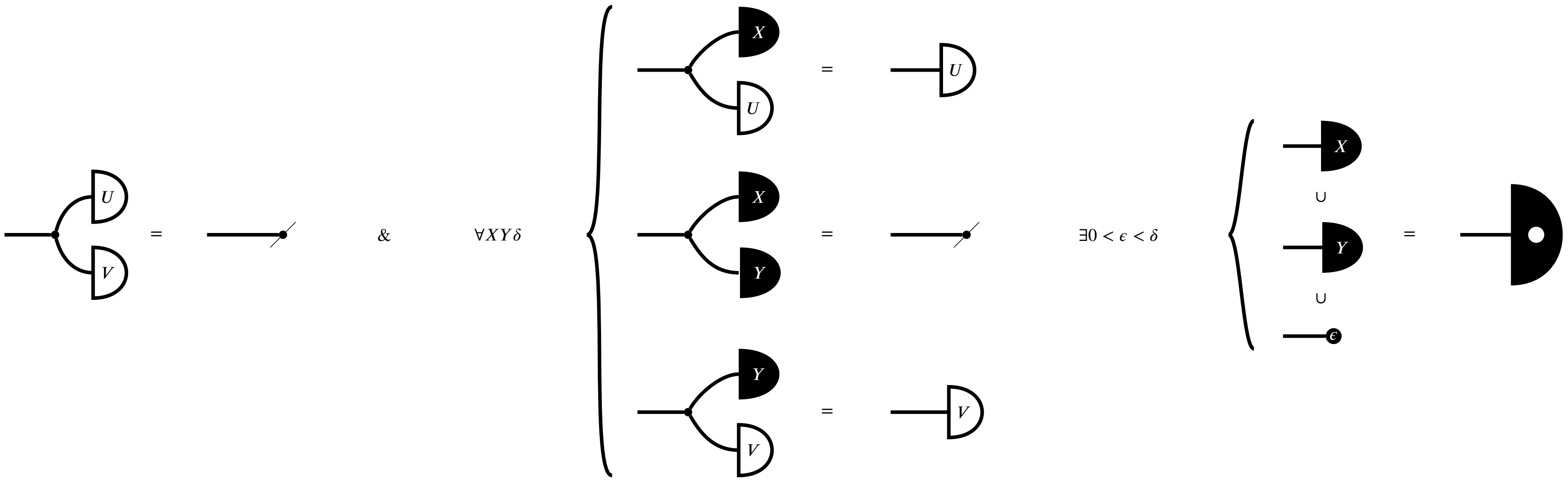
U is *a part of V*



Touching

Let's distinguish touching from overlap. Two shapes are "touching" intuitively when they are as close as they can be to each other, somewhere; any closer and they would overlap. Let's assume that we can restrict our attention to the parts of the shape that are touching, and that we can fill in the holes of these parts. At the point of touching, there is an infinitesimal gap – just as when we touch things in meatspace, there is a very small gap between us and the object due to the repulsive electromagnetic force between atoms. To deal with infinitesimals we borrow the $\epsilon - \delta$ trick from mathematical analysis; for any arbitrarily small δ, we can pick an even smaller ball of radius ϵ such that if we stick the ball in the gap, the ball forms a bridge that overlaps the two filled-in shapes, which allows us to draw a continuous line between them. Diagrammatically, this is:

U and *V* are *touching*

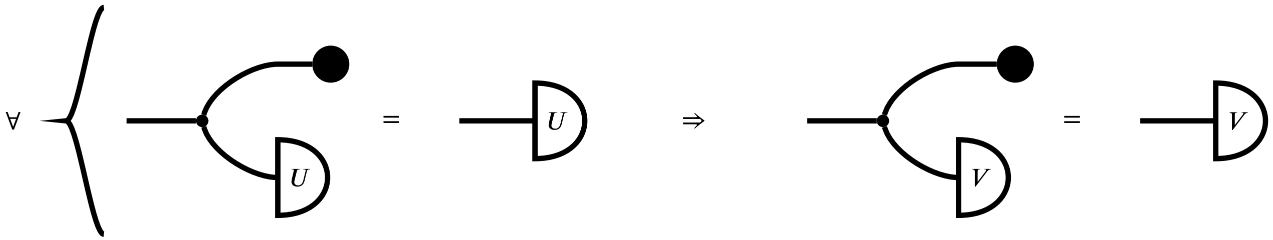


Within

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

U surrounds V

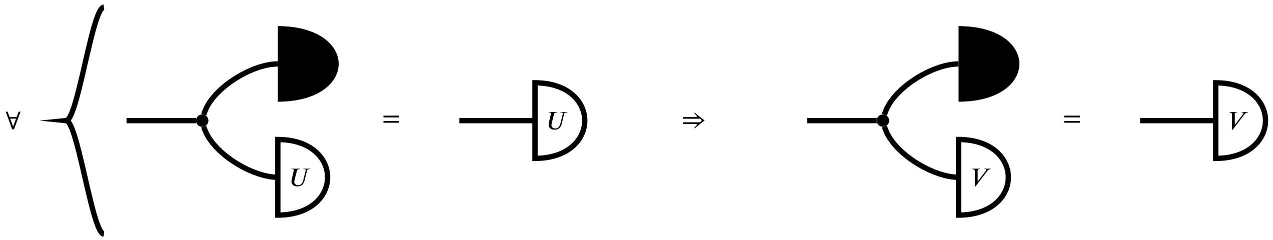
V is *within U*



Containers and enclosure

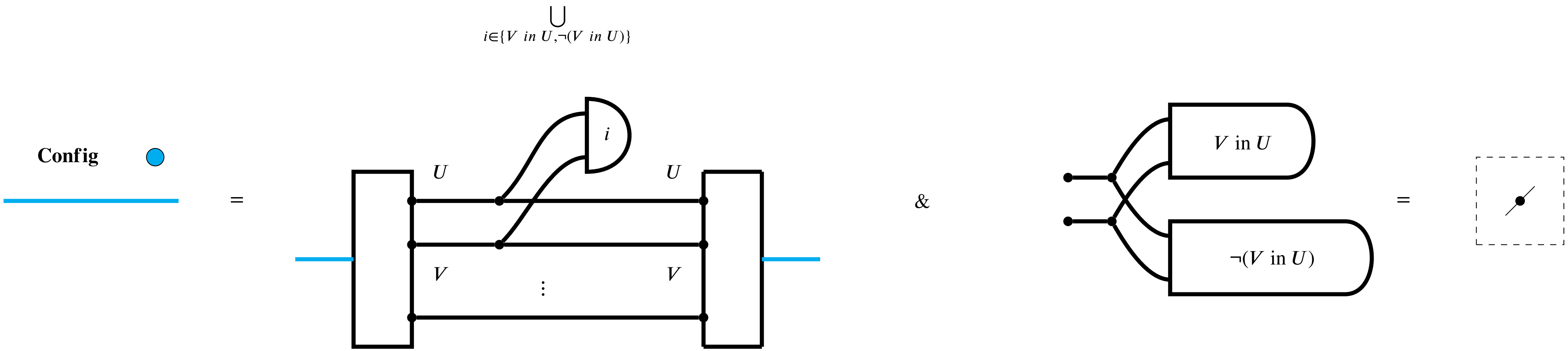
There is a strong version of within-ness, which we will call enclosure. As in when we are in elevators and the door is shut, nothing gets in or out of the container. Intuitively, there is a hole in the container surrounded on all sides, and the contained shape lives within the hole. To give a real-world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

U encloses V



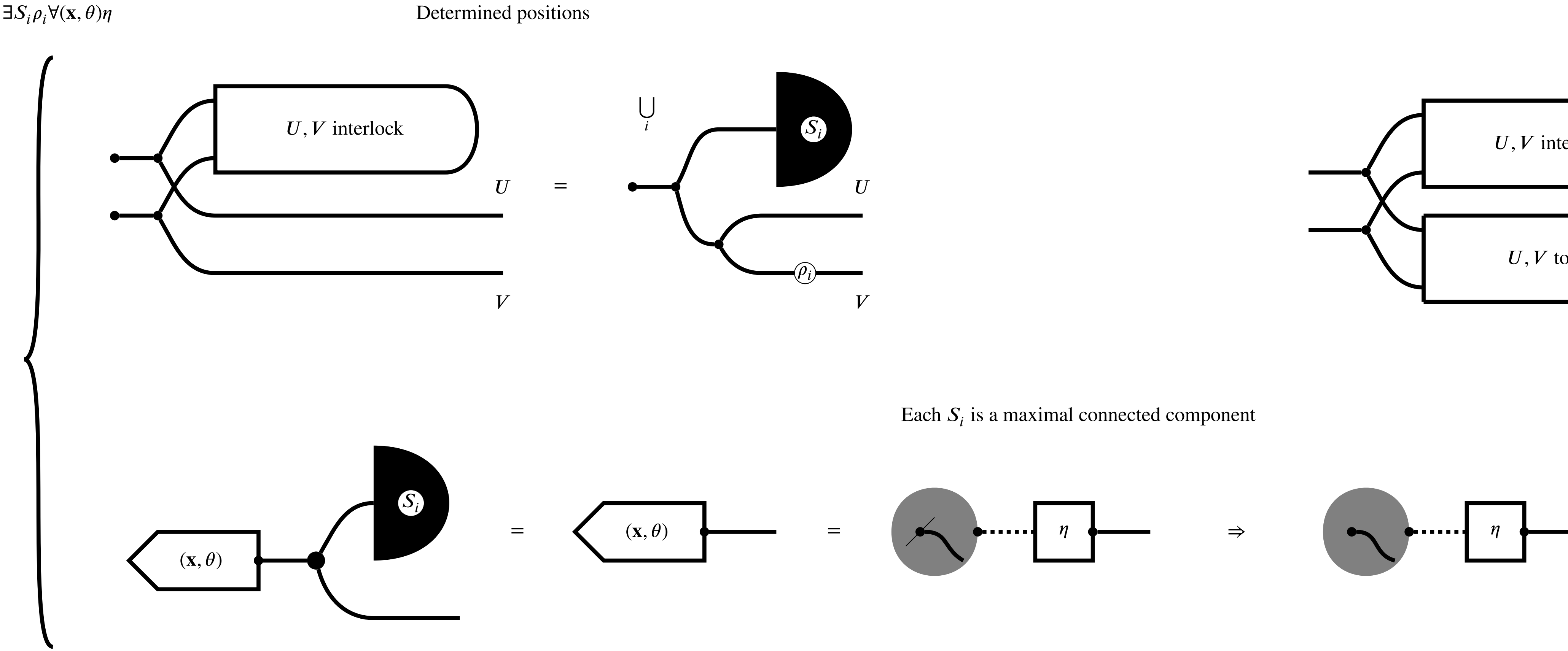
Trapped

There is an intermediate notion between within-ness and enclosure; for instance, standing in the stonehenge you are surrounded by the pillars, but you can always walk away, whereas if the pillars are very close, such as the bars of a jail cell, a human would not be able to leave the trap while still being able to see the outside. The difficulty here is that relative sizes come into play: small animals would still consider it a case of mere within-ness, because they can still walk away between the bars. So we would like to say that no matter how the pair of objects move rigidly, being trapped means that the trapped V stays within U. In other words, that in configuration space, if we forget about all other shapes, we can partition our space of configurations by two concepts, whether V is within U or not, and moreover that these two components is disjoint – i.e. not simply connected – so there is no rigid motion that can allow V to escape from being within U if V starts off trapped inside in U.



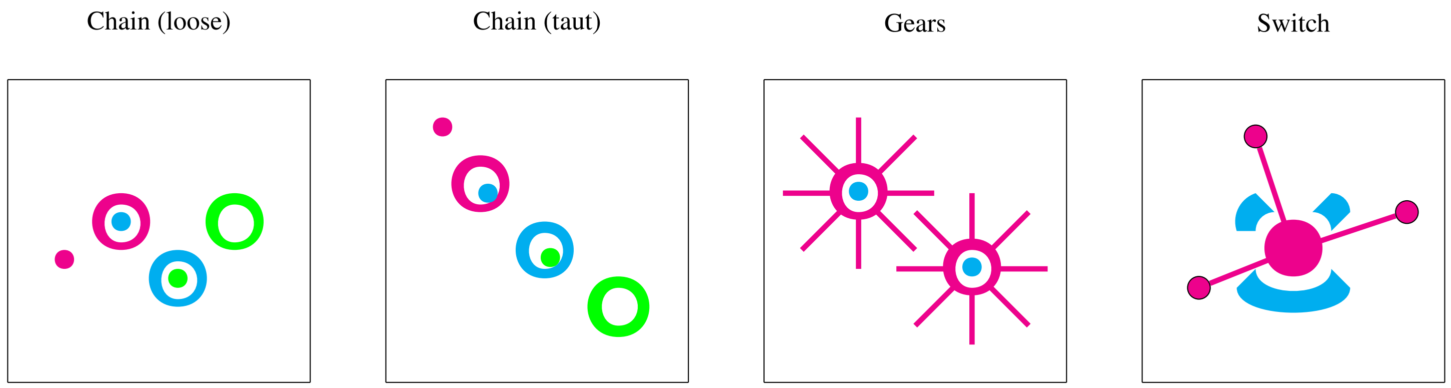
Interlocked

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



Constrained motion

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



Chain (loose)

Chain (taut)

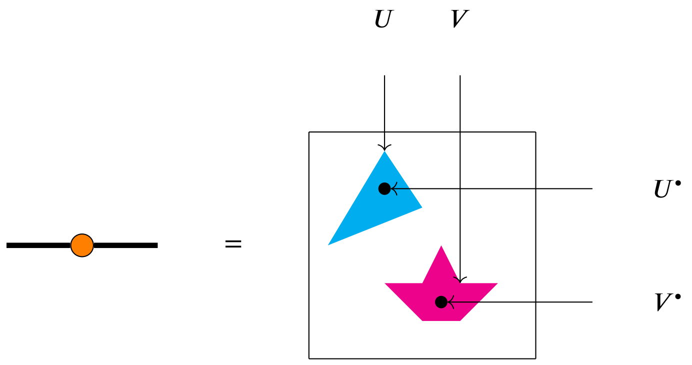
Gears

Switch

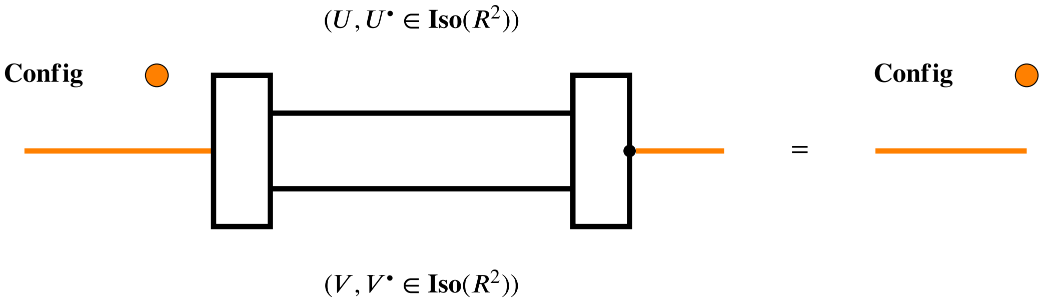
There are as many definitions to go through here as there are potential mechanical models, and among other things, there are mechanically realised clocks [], computers [], and analogues of electric circuits []. So instead, we will allow ourselves to additionally specify open sets as concepts in configuration space that correspond to whatever mechanical concepts we please, and we assure the reader seeking rigour that blueprints exist for all the mechanisms humans have built. Of course in reality mechanical motions are reversible among rigid objects, and directional behaviour is provided by a source of energy, such as gravitational potential, or wound springs. But we may in principle replace these sources of energy by a belt that we choose to spin in one direction – our own arrow of time. We postpone discussion of causal-mechanistic understanding and analogy for a later section.

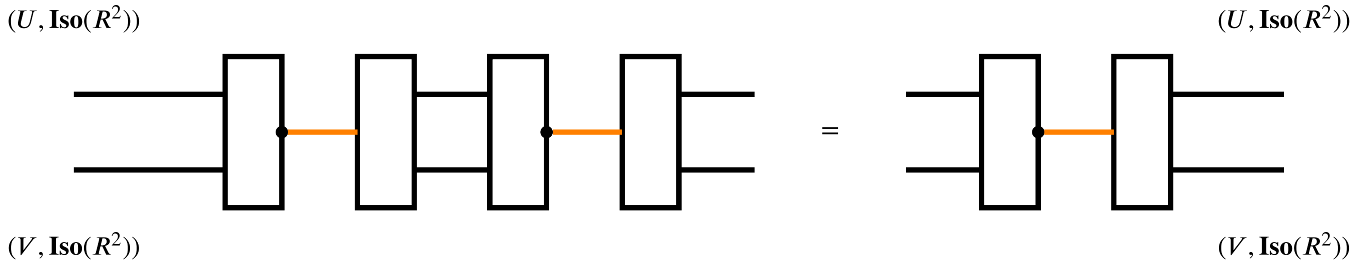
3.9.7 States, actions, manner

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

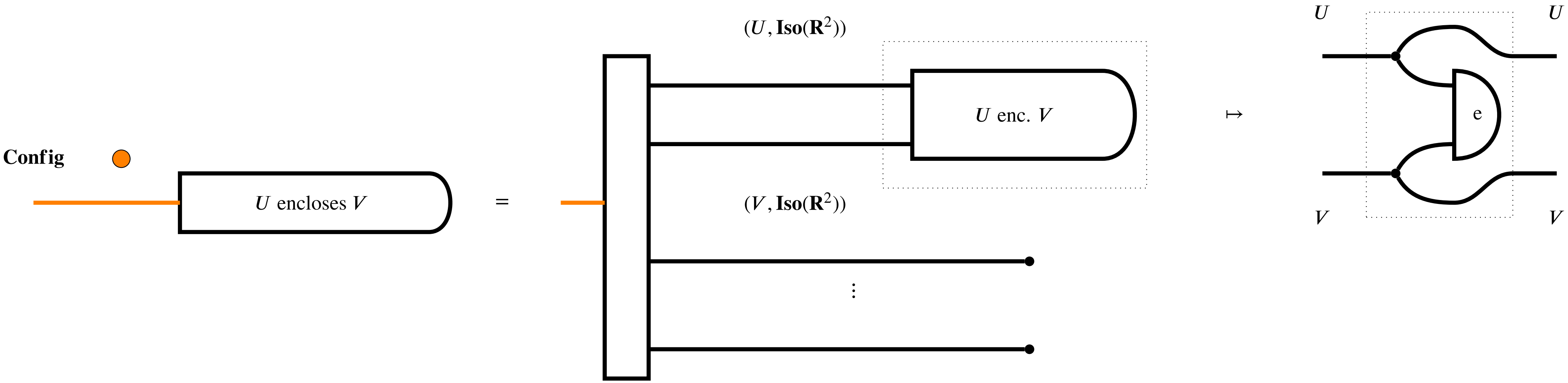


Concretely, for each shape in a sticky spider,
in order to reconstruct the shape,
the data of configuration space
only has to remember a basepoint
(which the isometries act upon)
paired with a label naming the shape
(so that the extent of the shape is reconstructible)

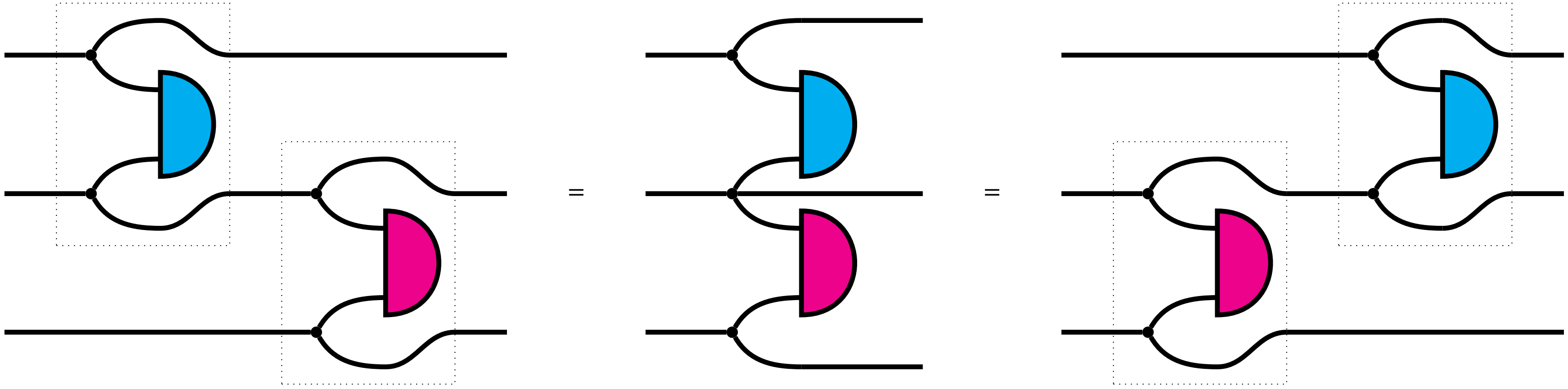

$(U, U^\bullet \in \mathbf{Iso}(R^2))$
Config
Config
$=$
$(V, V^\bullet \in \mathbf{Iso}(R^2))$



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

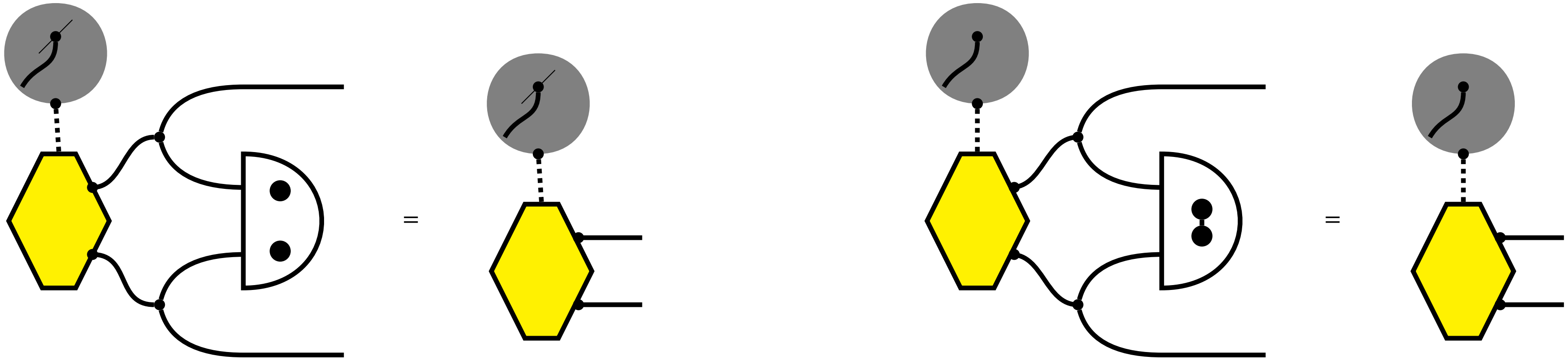

Config
U encloses V
=
$(U, \mathbf{Iso}(\mathbf{R}^2))$
U enc. V
$(V, \mathbf{Iso}(\mathbf{R}^2))$
⋮
↦
U
U
e
V
V

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



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

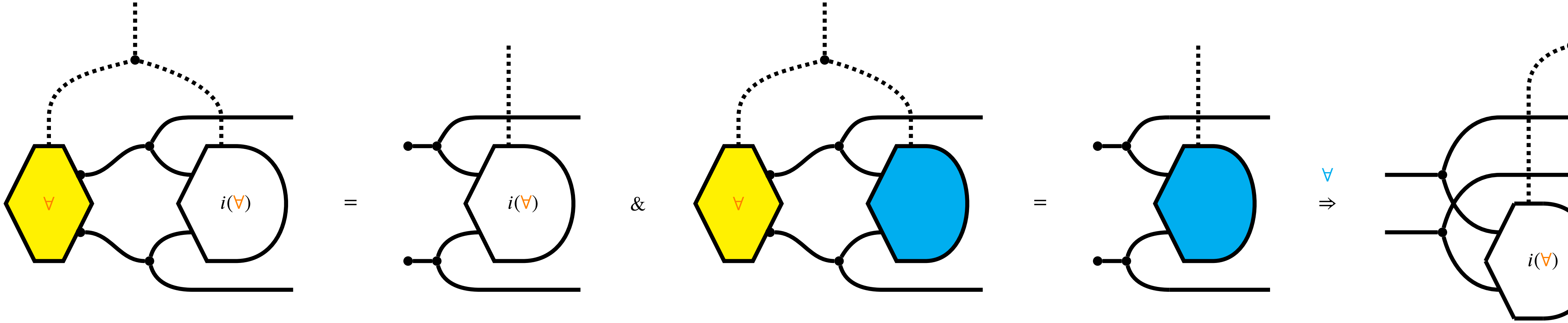
A particular collision trajectory



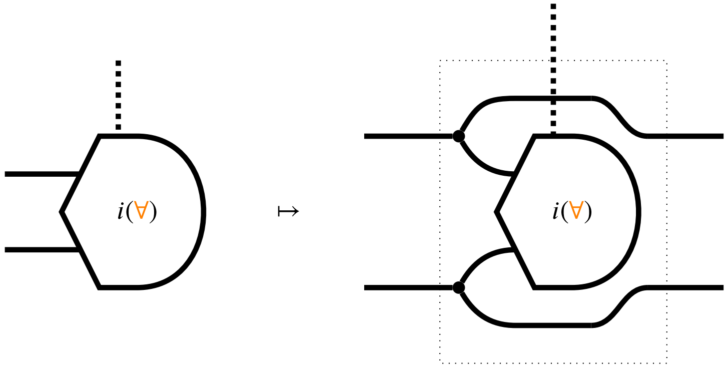
At $t = 0$, the shapes do not touch

At $t = 1$, the shapes touch

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

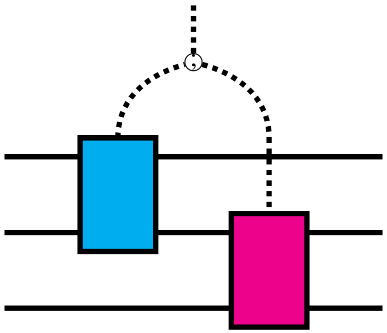


Once we have the open set $i(\forall)$ that corresponds to all expressible collisions, we have a homotopy-parameterised gate. Following a similar procedure, we can construct gates of motion that satisfy whatever pre- and post-conditions we like.

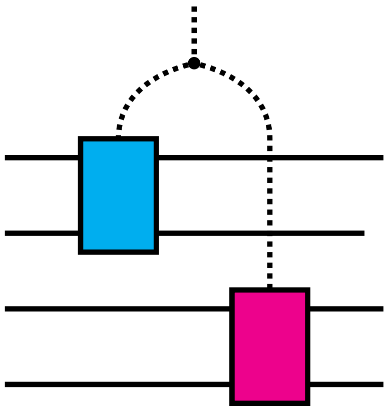


We can compose multiple rigid motions sequentially by a continuous function ; that splits a single unit interval into two: $; := x \mapsto \begin{cases} (2x, 0) \text{ if } x \in [0, \frac{1}{2}) \\ (1, 2x-1) \text{ if} x \in [\frac{1}{2}, 1] \end{cases}$. The effect of the map is to splice two vignettes of the same length together by doubling their speed, then placing them one after the other. We can achieve the same thing without resorting to units of measurement, because recall by Theorem 3.9.5 and by construction that we have access to a map that selects midpoints for us; we will revisit a string-diagrammatic treatment of homotopy and tenses in a later section. We can also compose multiple motions in parallel by copying the unit interval, allowing it to parameterise multiple gates simultaneously.

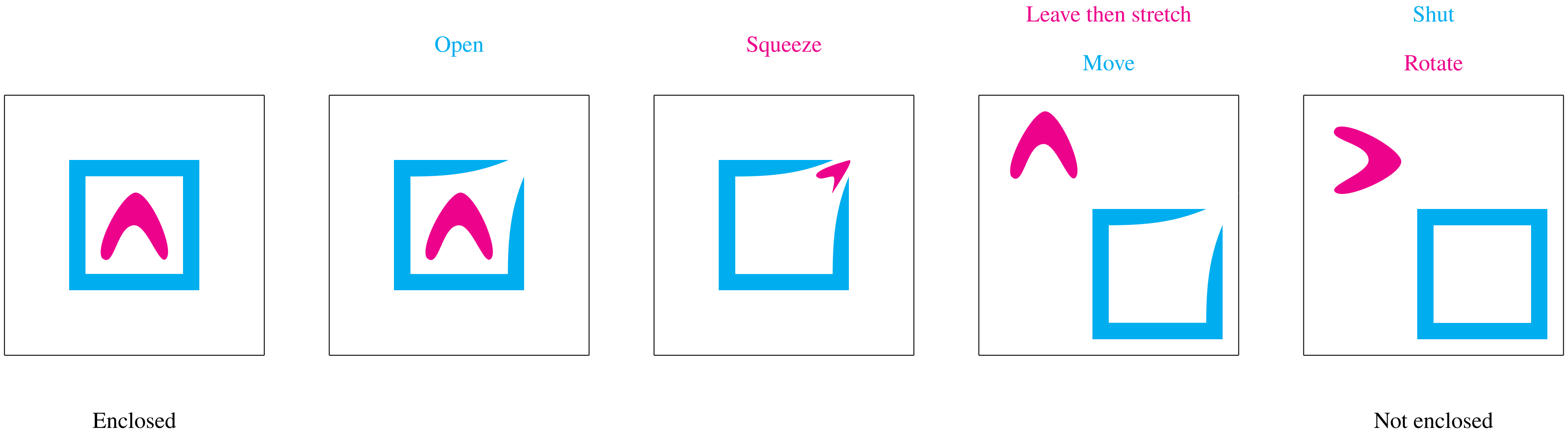
Sequential composition of motions



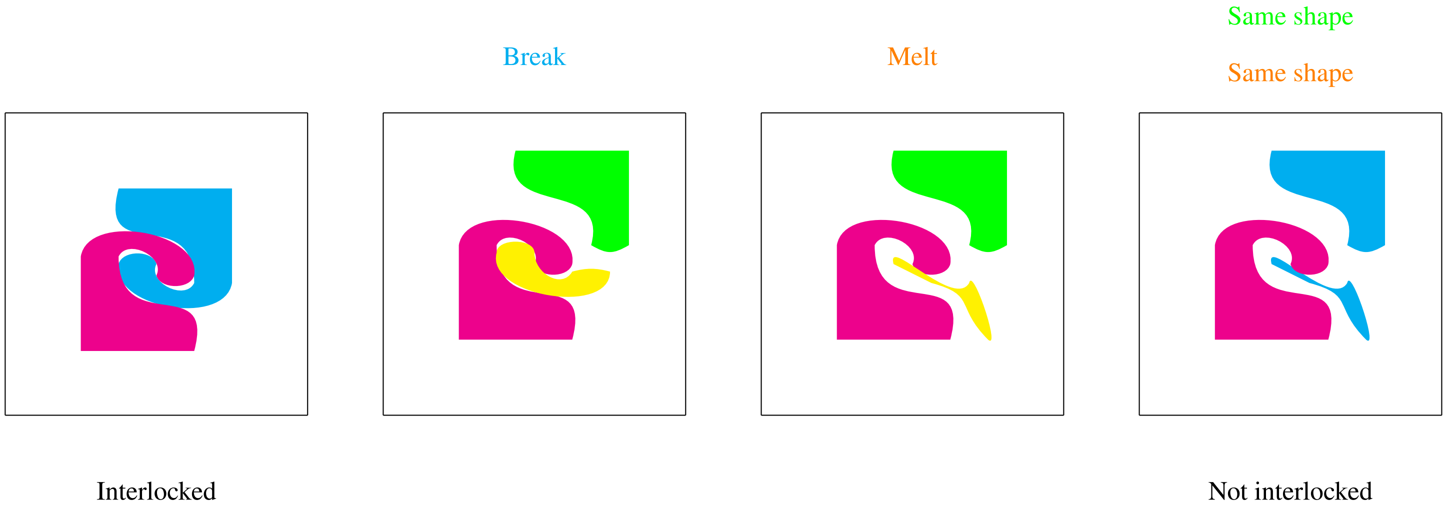
Parallel composition of motions



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



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

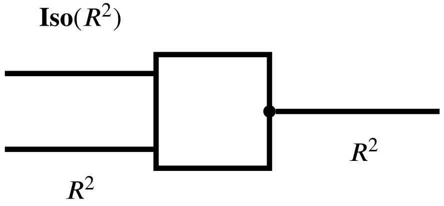


We provide the following construction as a general recipe to construct homotopies between spiders.

Construction 3.9.9 (Morphing sticky spiders with homotopies)**.** We aim to construct homotopies relating (almost) arbitrary sticky spiders. For now we focus on just changing one shape into another arbitrary one. The idea is as follows. First, we need a cover of open balls $\cup\mathcal{J} = T^0$ and $\cup\mathcal{K} = T^1$ of the start and end cores T^0 and T^1 such that each $k \in T^1$ is expressible as a rigid isometry of some core $j \in \mathcal{J}$; this is so we can slide and rearrange open balls comprising T^0 and reconstruct them as T^1. As an intermediate step to eliminate holes and unify connected components, we gather all of the balls at a meeting point m (to be determined shortly.) Intuitively we can illustrate this process as follows:

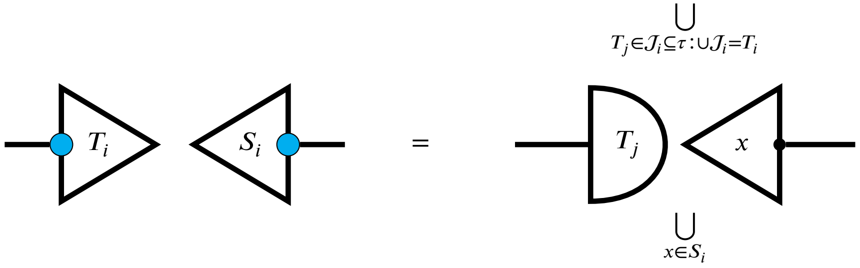
placeholder

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



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

Third, before we begin sliding the open balls, we must ensure that the halo of the shape cooperates. We observe that a given shape i in a sticky spider may be expressed as the union of a family of constant continuous partial functions in the following way. Given an open cover $\mathcal{J}$ such that $\cup\mathcal{J} = T_i$, where T_i is the core of the shape i, each function is a constant map from some $T_j \in \mathcal{J}$ to some point $x \in S_i$, where S_i is the halo of the shape i. For each $T_j \in \mathcal{J}$ and every point $x \in S_i$, the constant partial function that maps T_j to x is in the family.


T_i
S_i
$=$
$\bigcup_{T_j \in \mathcal{J}_i \subseteq \tau : \cup\mathcal{J}_i = T_i}$
T_j
x
$\bigcup_{x \in S_i}$

By definition of sticky spiders, there must exist some point m that is in both the core and the halo: we pick such a point as the rendezvous for the open balls. For each partial map in the family, we provide a homotopy that varies only the image point x continuously in the space to finish at m. In the following illustration, we colour the halo in yellow, the core in cyan, and their overlap in green.

placeholder

Now we can slide the open balls to the rendezvous m. Since homotopies are reversible by the continuous map $t \mapsto (1 - t)$ on the interval, we can perform the above steps for shapes T^0 and T^1 to finish at the same open ball, reversing the process for T^1 and composing sequentially to obtain a finished transformation.

$$placeholder$$

The final wrinkle to address is when dealing with multiple shapes. Recalling our exclusion conditions **??** for shapes, it may be that parts of one shape are enclosed in another, so the processes must be coordinated so that there are no overlaps. For example, the enclosing shape must be first opened, so that the enclosed shape may leave.

$$placeholder$$

I struggle to come up with a proof that all spiders $\mathbf{R}^2$ are mutually transformable by homotopy in this (or any other) way, so that will remain a conjecture. But it is clear that a great deal of spiders are mutually transformable; almost certainly any we would care to draw. So this will just be a construction for now.

Concretely, we do this by sandwiching the verb gate between the section of the configuration of one spider and the retract of the configuration space of another spider.

$$sandwich$$

3.10 Continuous Relations: A model setting for text circuits

In this section, we introduce *continuous relations*, which are a naïve (and to the best of my knowledge, unstudied) extension of the category **Top** of topological spaces and continuous functions towards continuous relations. We choose this category (as opposed to plain **Rel**, the category of sets and relations) because it satisfies several requirements, which we list below along with the justifications by which we obtained them. The justifications involve basic reflections upon language use, and the consequent mathematical constraints those affordances impose on any interpretation of text circuits in a symmetric monoidal category. A priori it could well be that there is no non-trivial process theory that satisfies these constraints, so the onus is on us to show that there exists such a process theory. We demonstrate at the end of this chapter that the category **TopRel** meets these requirements.

3.10.1 Justification 1: bookwork demonstration

First, and simplest, is so that we have an excuse to build up a symmetric monoidal category from scratch, just to see what formal work is involved in doing so.

3.10.2 Justification 2: anything can be a noun-wire

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

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

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

$$placeholder$$

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

$$placeholder$$

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

$$placeholder$$

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

$$placeholder$$

So in summary, we are asking for the following:

Requirement 3.10.1.

3.10.3 Justification 3: we need both discrete symbolic and spatial conceptual behaviours

Third, we have a cognitive consideration: how do we move between concepts – however they are represented – and symbolic representation and manipulation? Here we sidestep the debate around what concepts are, aligning ourselves close to Gärdenfors: we assume that there are *conceptual spaces* that organise concepts of similar domain, and that regions of these spaces correspond to concepts. Gärdenfors' stance is backed by empirical data [], but even if he is wrong, he is at least interestingly so for our purposes.

The classic example of colour space is also one of the best studied and implemented: there are many different embeddings of the space of visible colours in Euclidean space []. In this setting we can mathematically model the action of categorising a particular point in colour space as `blue` by checking to see whether that point falls within the region in colourspace that the symbol `blue` is associated with. So we find that this view is conducive to modelling concepts as spatial entities – a very permissive and expressive framework, which will allow us to calculate interesting things – whilst also having the ability to handle them using symbolic labels.

But once we have a stock of symbols referring to entities in space, we can start talking about pairs of entities (e.g. `red or blue`), subsets of sets of entities (e.g. `autumnal colours`), arbitrary relations between the set of entities in one space and entities of another (e.g. how the colour of a banana relates to its probable textures and tastes). Given enough time and patience, we can linguistically construct – at least – any finite relation between finite sets. So we are asking for the following:

Requirement 3.10.2.

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

3.10.4 Justification 4: topology is a good setting to model concepts spatially

Where we differ from Gärdenfors' is that we only ask for topological spaces, rather than the stronger notion of metric spaces. There are several reasons for this choice. First, topology is a basic mathematical framework for space. Second, we can view topology as a framework for conceptual spaces, where we consider the open sets of a topology to be the concepts. On this latter point, Éscardo provides a the following correspondence [], which we extend with "Point" and "Subset", and an additional column for interpretation as conceptual space:

In **TopRel**, open sets are precisely tests. Modelling concepts as open sets or tests aligns them with semi-decidability. Given any state-instance, we can test whether it overlaps with a concept graphically: success returns a unit scalar, failure returns the zero scalar.

4

Sketches of the shape of language

4.1 Modelling metaphor

I will take a *metaphor* to be text where systematic language for one kind of concept is used to structure meanings for another kind of concept where literal meanings cannot apply. This may subsume some cases of what would otherwise be called *similes* or *analogies*.

THE IDEA: First, we observe that we can model certain kinds of analogies between conceptual spaces by considering structure-preserving maps between them. For example, Planck's law gives a partial continuous function from part of the positive reals measuring Kelvin to wavelengths of light, and the restriction of this mapping to the visible spectrum gives the so-called "colour temperature" framework used by colourists.

Second, we observe that we can use simple natural language to describe conceptual spaces, instead of geometric or topological models. Back to the example of colour temperature, instead of precise values in Kelvin, we may instead speak of landmark regions that represent both temperature and colour such as "candle", "incandescent", "daylight", which obey both temperature-relations (e.g. candle is a lower temperature than incandescent) and colour-relations (daylight is bluer than incandescent).

Third, we observe that we can also use simple natural language to describe more complex conceptual schemes with interacting agents, roles, objects, and abilities. Organising this linguistic data in the concrete structure of a text circuit allows us to formally specify what it means for one conceptual scheme to structure another by describing structure-preserving maps between the text circuits. This will allow us to reason about and calculate with metaphors such as "Time is Money".

THE MOTIVATION:

4.1.1 Orders, Temperature, Colour, Mood

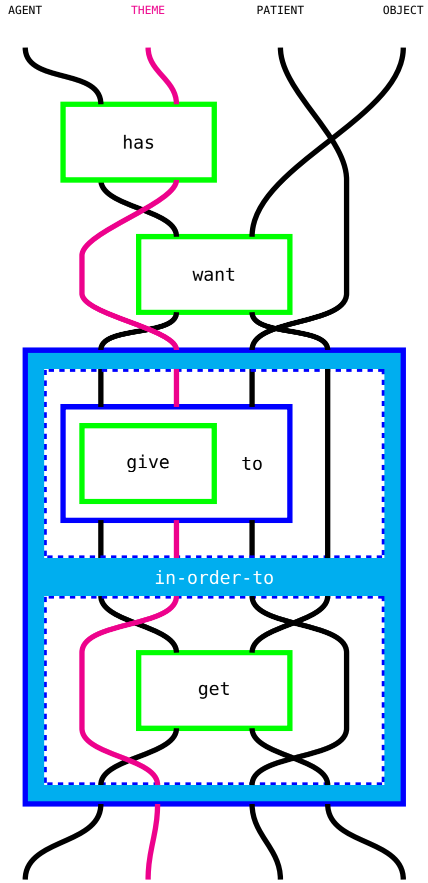
4.1.2 Complex conceptual structure

TIME IS MONEY

```
"Do you have time to look at this?"
"This is definitely worth the time!"
"What a waste of time."
```

I will work through an example of partially using the concept of Money to structure that of Time. Part of the concept of Money is that it can be *exchanged* for something. The concept of exchange can be glossed approximately as the following text, with variable noun-entries capitalised.

```
AGENT has THEME.
AGENT wants OBJECT.
AGENT gives THEME to PATIENT in-order-to get OBJECT.
```



The

4.1.3