

VINCENT WANG-MAŚCIANICA

STRING DIAGRAMS FOR TEXT

Contents

<i>0</i>	<i>Introduction and overview</i>	<i>5</i>
<i>0.1</i>	<i>What is this thesis about?</i>	<i>6</i>
<i>0.2</i>	<i>Whose shoulders, and how high?</i>	<i>8</i>
<i>0.3</i>	<i>How do I work with this?</i>	<i>13</i>
<i>0.4</i>	<i>What new things can we do?</i>	<i>13</i>
<i>0.5</i>	<i>Other questions, such as:</i>	<i>13</i>
<i>0.5.1</i>	<i>Why is this thesis landscape?</i>	<i>13</i>
<i>0.5.2</i>	<i>What is this thesis not about?</i>	<i>13</i>
<i>0.5.3</i>	<i>Why should I read this?</i>	<i>14</i>

<i>1</i>	<i>String diagrams</i>	<i>17</i>
<i>1.1</i>	<i>A Partial History of String Diagrams</i>	<i>17</i>
<i>1.1.1</i>	<i>Pre-formal Diagrams</i>	<i>17</i>
<i>1.1.2</i>	<i>Convergent Evolution</i>	<i>18</i>

<i>1.2</i>	<i>Process Theories</i>	<i>19</i>
<i>1.2.1</i>	<i>What does it mean to copy and delete?</i>	<i>22</i>
<i>1.2.2</i>	<i>What is an update?</i>	<i>24</i>
<i>1.2.3</i>	<i>Spatial predicates</i>	<i>25</i>
<i>1.2.4</i>	<i>Some basic properties of linguistic spatial relations</i>	<i>26</i>
<i>1.3</i>	<i>Defining String Diagrams</i>	<i>27</i>
<i>1.3.1</i>	<i>Symmetric Monoidal Categories</i>	<i>27</i>
<i>1.3.2</i>	<i>PROPs</i>	<i>27</i>
<i>1.3.3</i>	<i>1-object 4-categories</i>	<i>27</i>
<i>1.4</i>	<i>What did Montague think "grammar" was?</i>	<i>28</i>
<i>1.4.1</i>	<i>Historical Context and Scope of Analysis</i>	<i>28</i>
<i>1.4.2</i>	<i>On "Background Notions"</i>	<i>29</i>
<i>1.4.3</i>	<i>On Syntax</i>	<i>30</i>

<i>2</i>	<i>Text circuits</i>	<i>33</i>
<i>2.1</i>	<i>How do we communicate?</i>	<i>34</i>
<i>2.1.1</i>	<i>Speaking grammars, listening grammars</i>	<i>34</i>
<i>2.1.2</i>	<i>A context free grammar to generate Alice sees Bob quickly run to school</i>	
<i>2.1.3</i>	<i>Relating the generative grammar to a pregroup grammar via a discrete monoidal fib</i>	
<i>2.1.4</i>	<i>Discrete Monoidal Fibrations</i>	<i>47</i>
<i>2.1.5</i>	<i>Discrete monoidal fibrations for grammatical functions</i>	<i>52</i>
<i>2.1.6</i>	<i>Extended analysis: Tree Adjoining Grammars</i>	<i>52</i>

3	<i>Continuous relations</i>	59	
3.1	<i>Continuous Relations</i>	60	
3.2	<i>Continuous Relations by examples</i>	61	
3.3	<i>The category TopRel</i>	67	
3.4	<i>Biproducts</i>	67	
3.5	<i>Symmetric Monoidal Closed structure</i>	68	
3.6	<i>TopRel diagrammatically</i>	69	
3.6.1	<i>Relations that are always continuous</i>	69	
3.7	<i>Populating space with shapes using sticky spiders</i>	73	
3.7.1	<i>When does an object have a spider (or something close to one)?</i>	73	
3.8	<i>Sticky spiders are collections of shapes in space</i>	93	
3.9	<i>Displacing shapes with symmetries</i>	93	
3.10	<i>Moving shapes continuously</i>	93	
3.11	<i>Configuration space of a sticky spider</i>	93	
3.12	<i>Topological models for a selection of concepts</i>	93	
3.13	<i>Modelling Lassos</i>	94	
4	<i>Metaphor, narrative, other pictures</i>	105	
4.1	<i>Modelling metaphor</i>	106	
4.1.1	<i>Orders, Temperature, Colour, Mood</i>	106	
4.1.2	<i>Complex conceptual structure</i>	106	
4.1.3		107	

0

Introduction and overview

0.1 *What is this thesis about?*

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

THE FORMAL DIAGRAMS ARE 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.

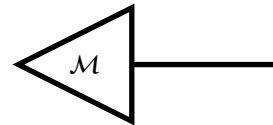
It turns big black boxes into composites of small black boxes,
which leaves smaller pieces for machines to learn representations for.

We introduce the history, theory, and use of these diagrams in Chapter 1.

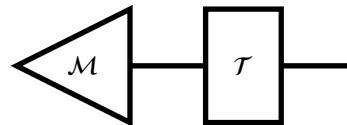
THE DIAGRAMS LOOK LIKE CIRCUITS.

Let's say that the meaning of text is how it updates a model.

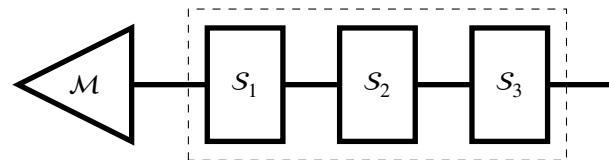
So we start with some model of the way things are.



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.

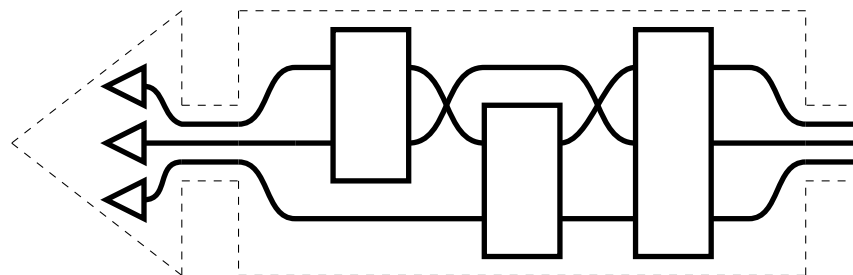


Let's say that *The meaning of a sentence is how it updates the meanings of its parts.*

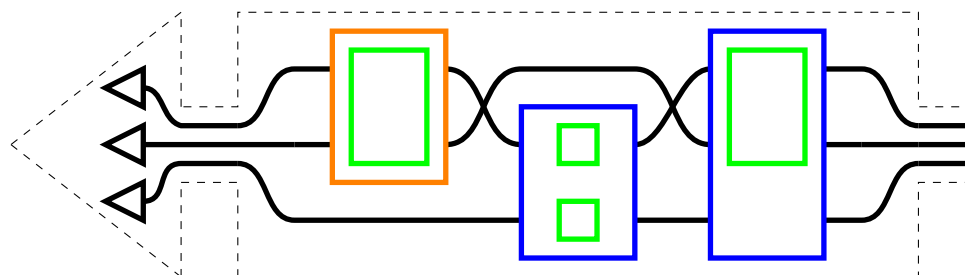
Let's say that the *parts* of a sentence are the nouns it contains or refers to.

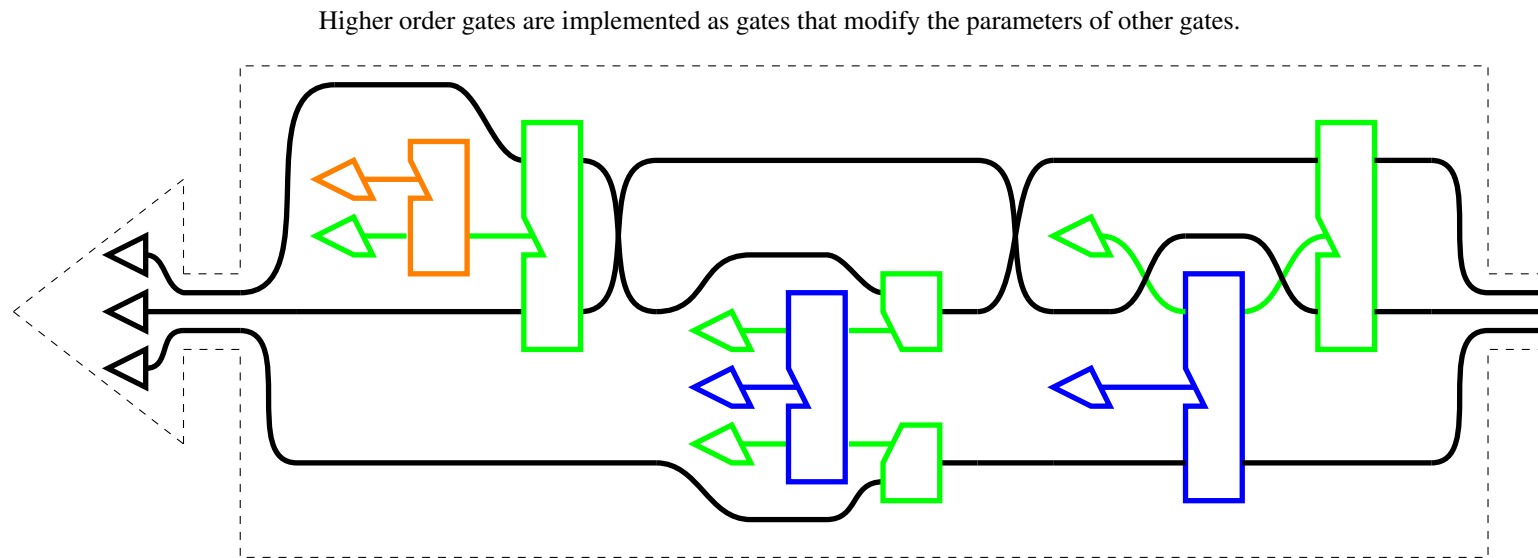
Noun data is carried by wires.

Collections of nouns are related by gates.



Gates can be related by higher order gates.





0.2 Whose shoulders, and how high?

THESE DIAGRAMS MAKE DIFFERENT WAYS OF THINKING ABOUT LANGUAGE LOOK THE SAME

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, you may represent text as circuits.

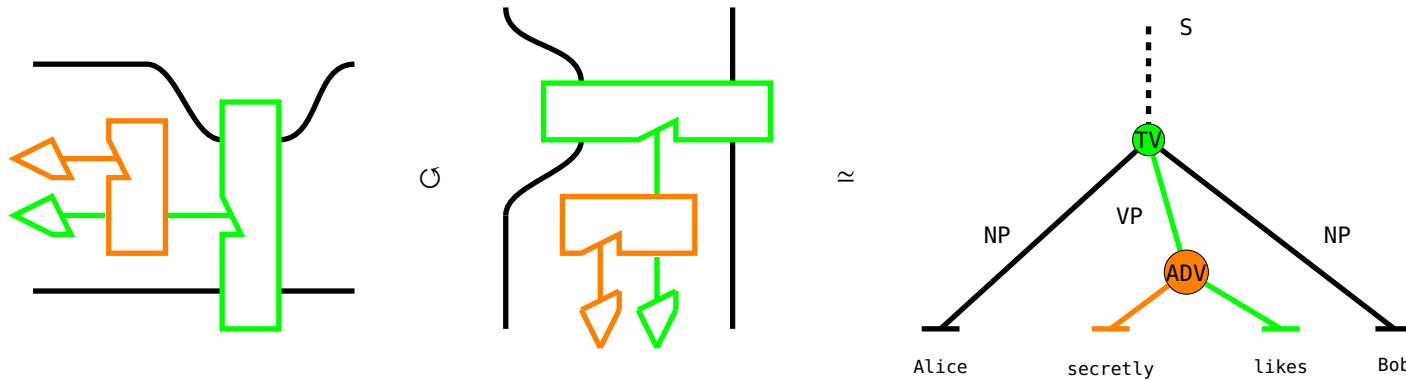
There are several ways to obtain gates from grammars for sentences.

CONTEXT-FREE GRAMMAR

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 GRAMMAR

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

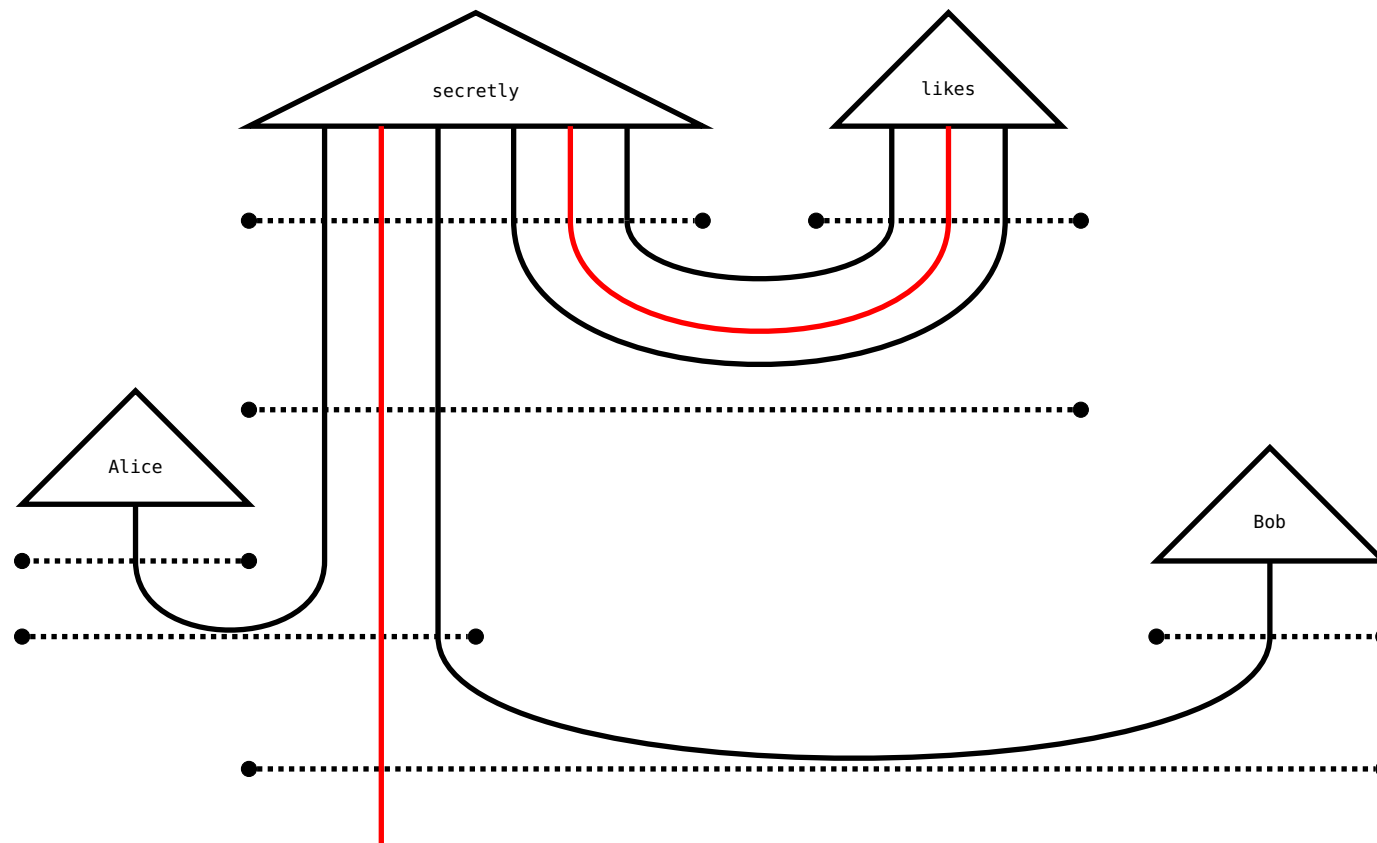
The latter is the grammar of the speaker: a full sentence is produced from S .

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

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

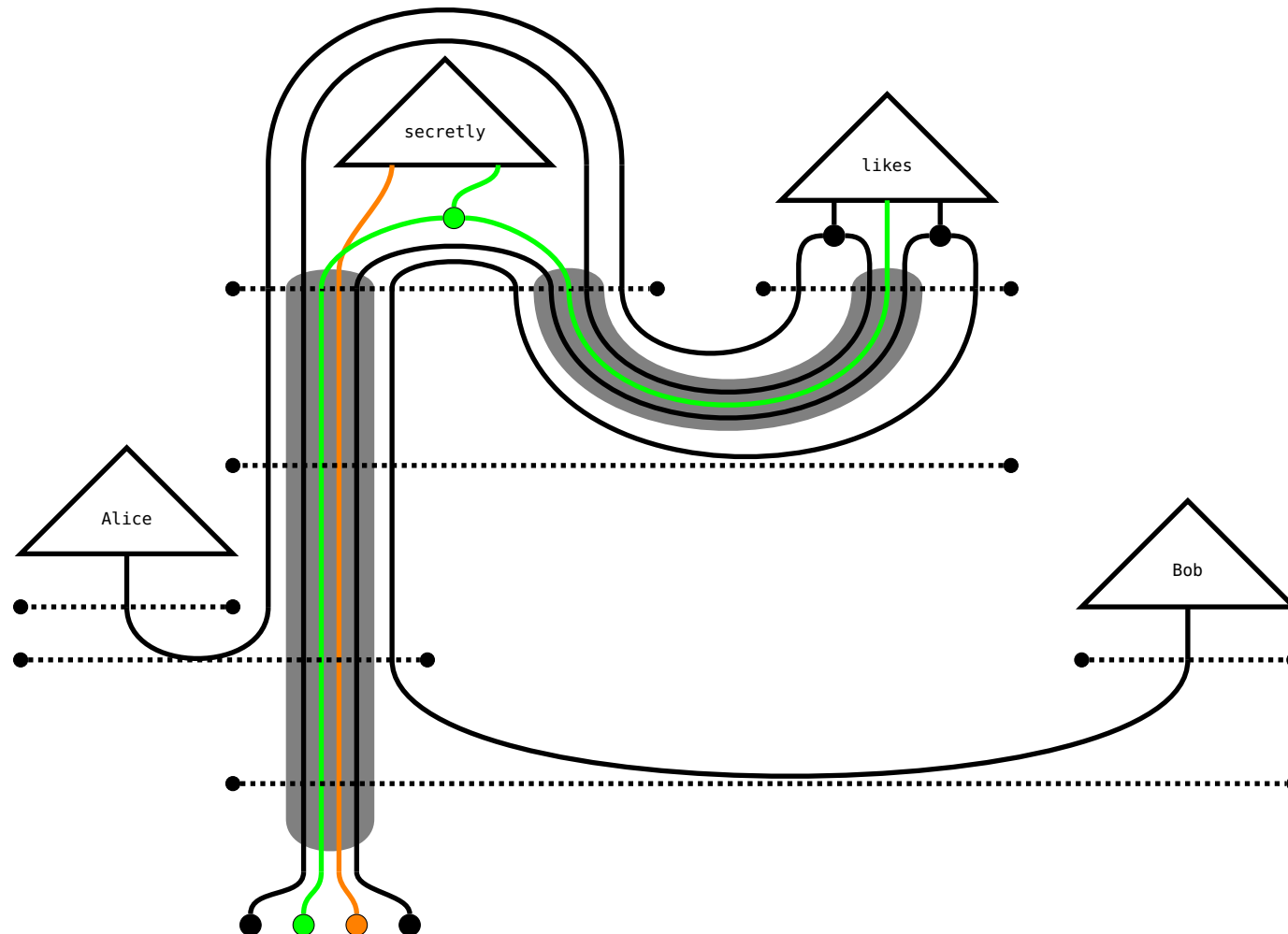
$$\begin{array}{c}
 \text{secretly} : (-^1 n \cdot s \cdot n^{-1}) \cdot (-^1 n \cdot s \cdot n^{-1})^{-1} \quad \text{likes} : ^{-1} n \cdot s \cdot n^{-1} \\
 \hline
 \text{Alice} : n \quad \text{secretly_likes} : ^{-1} n \cdot s \cdot n^{-1} \\
 \hline
 \text{Alice_secretly_likes} : s \cdot n^{-1} \quad [x^{-1} \cdot x \rightarrow 1] \quad \text{Bob} : n \\
 \hline
 \text{Alice_secretly_likes_Bob} : s \quad [x \cdot ^{-1} x \rightarrow 1]
 \end{array}$$

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

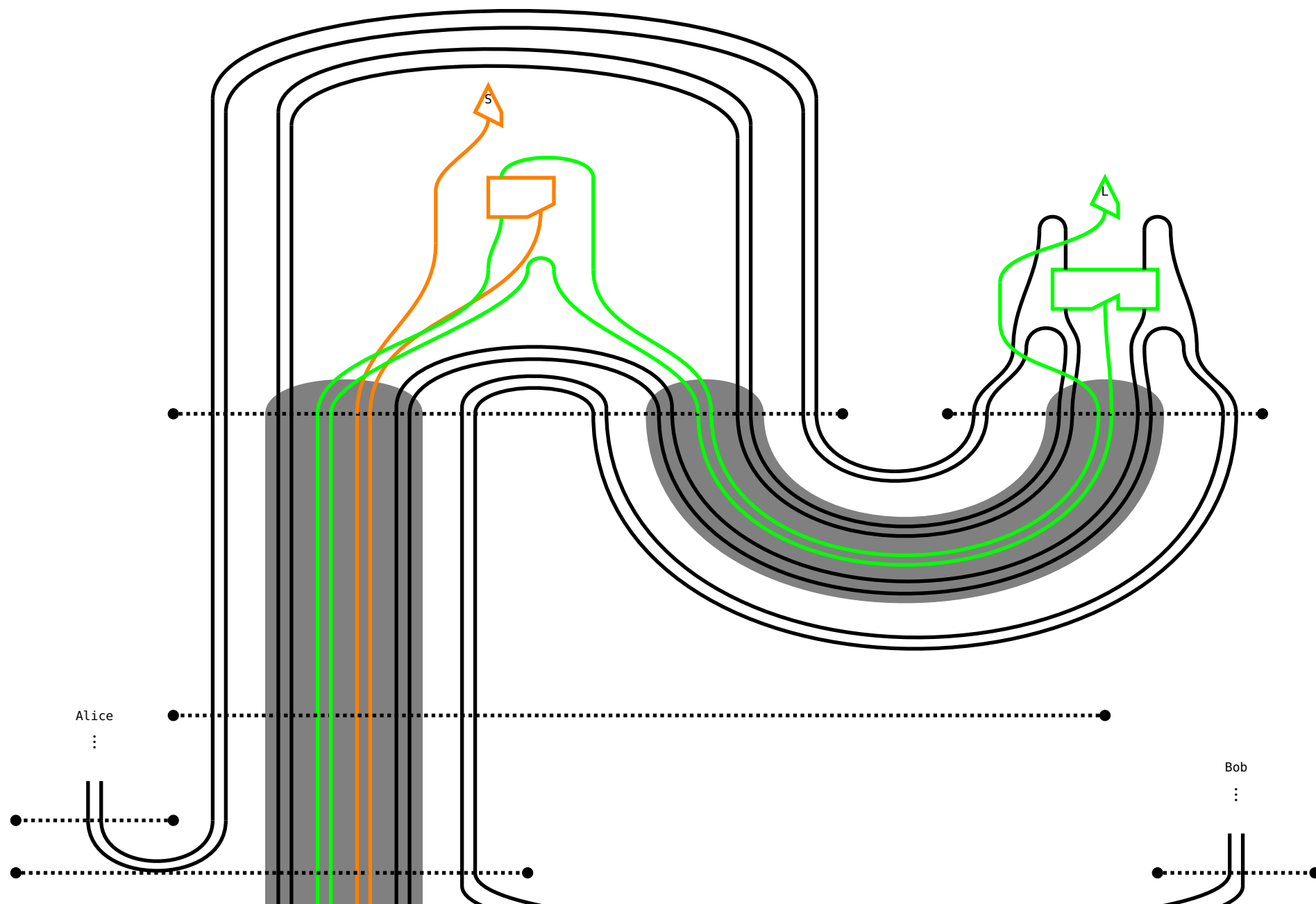


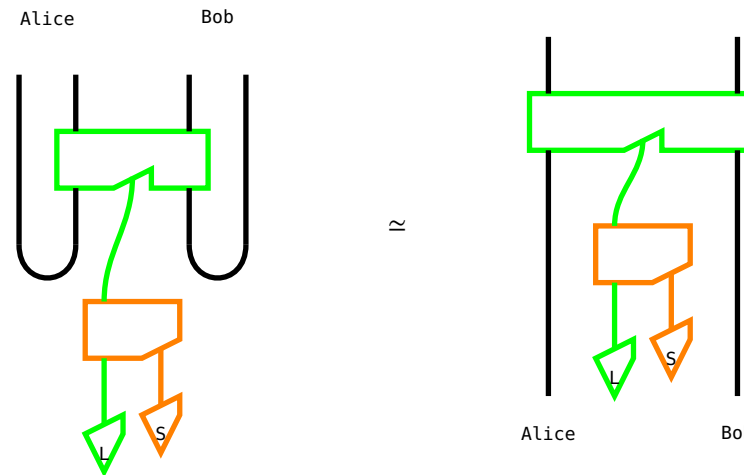
To obtain a circuit, we make the sentence type s dependent on the words that occur in the sentence, and we refine the structure on each word by introducing "internal wirings",

which make use of multiwires called spiders.



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





In Chapter ??, we show how certain pregroup and dependency grammars may be equipped with internal wirings to obtain text circuits.

0.3 *How do I work with this?*

0.4 *What new things can we do?*

0.5 *Other questions, such as:*

0.5.1 *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.

0.5.2 *What is this thesis not about?*

Although this thesis is about language, it is not really linguistics. Linguistics, conventionally, is about understanding human language. Formal linguistic theories rely on empirical observation of human language use. Now there are not-human entities that use human language. Now there are un-natural languages that humans use. So I am interested in sketching a form of language that humans and computers can use. This sketch is informed by, but not bound to, human language.

Although this thesis is mathematical, it is not really mathematics. This thesis does not introduce significantly new mathematical constructions or relations. So there are no new "know that"s. This thesis does use relatively

modern mathematics to approach an old problem. The math is symmetric monoidal categories, the problem is depicting language. So there is new "know how".

This thesis is computer science, a little. The mathematics used is, to a degree, implementation agnostic. This thesis is only concerned with the "in principle", rather than the "in practice". There will be no code demonstration nor machine learning experiments, because Computer Science is to Programming as Physics is to Engineering. I will point out where experiments have been done. Hypothetical procedures will be spelled out if needed.

0.5.3 *Why should I read this?*

If you are a linguist or a computer scientist: 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 language¹.

The presence of LLMs should not totally discourage our endeavour to understand language, but their sheer ability presents strong constraints about the acceptability of theories of language. Theories are idealised, partial, and frictionless. In constast, 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 theories². So new standards of quality or acceptability are needed, for instance:

- **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, you don't have a theory of language, you have a practical nonstarter.
- **If you can't beat them, be simpler.** There is value in a theory that provides understanding even if it is simplified, practically yet unrealised, or momentarily empirically embarrassed. But if your theory of language requires a comparable investment of formal education to understand as it takes to follow a machine learning tutorial online, you don't have a theory of language, you have a sociological dead-end.
- **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, you don't have a theory of language, you have a bore.
- **If you can't beat them, smile and look pretty.** There is value in beautiful things, and 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.

¹ 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?

² Just as the Ptolemaic geocentric model of the solar system was more empirically precise than the heliocentric Copernican, though the latter was closer to "correct understanding". Ptolemaic epicycles can approximate any observed trajectory of planets in a geocentric sky by fourier transform, and this overfitting is good if you are, say, a navigator. It took until Einstein's relativity to explain the perihelion of mercury, which at last aligned theoretical understanding with empirical observation.

- **You can't just ignore them.**

I claim to meet these standards. I present *ab initio* theory of language that is explicitly compatible with neural methods – Chapter ??, that consists of simple and formal diagrams – Chapter ?? – far prettier than their predecessors – Chapter ??. My diagrams unify the coalgebraic and algebraic views of syntax – Chapter ?? – and they can sing stories of space – Chapter ?? – and make merry metaphor – ??.

1

String diagrams

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.

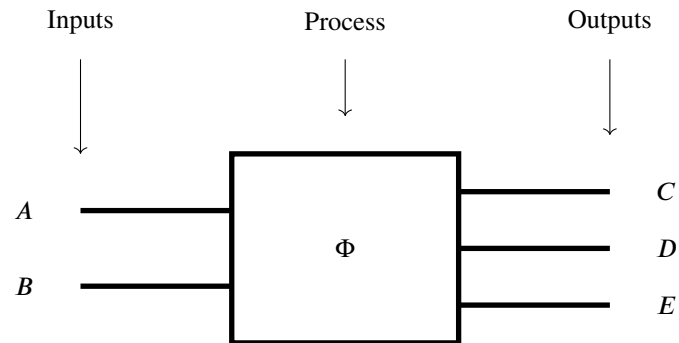
IN PHYSICS

IN LOGIC

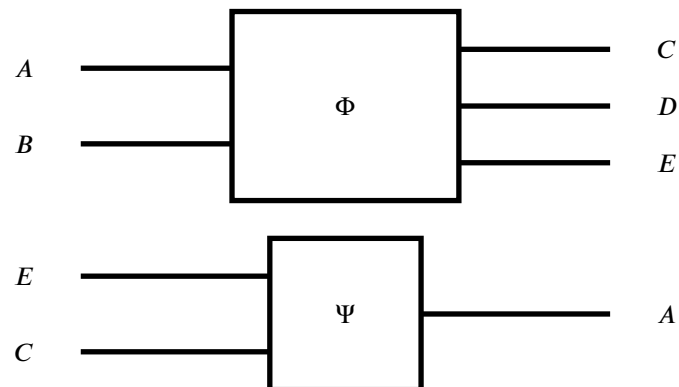
IN COMPUTER SCIENCE

1.2 Process Theories

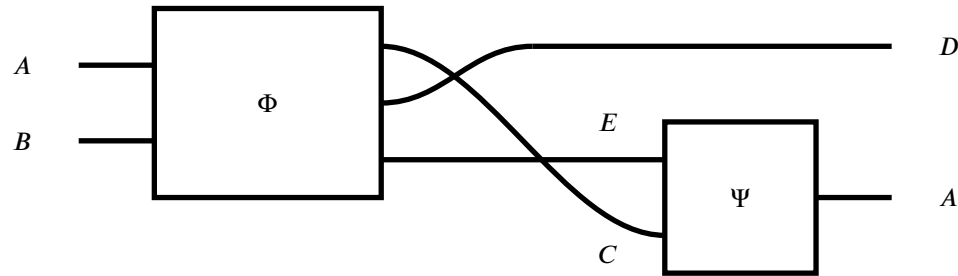
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. We read processes from left to right.



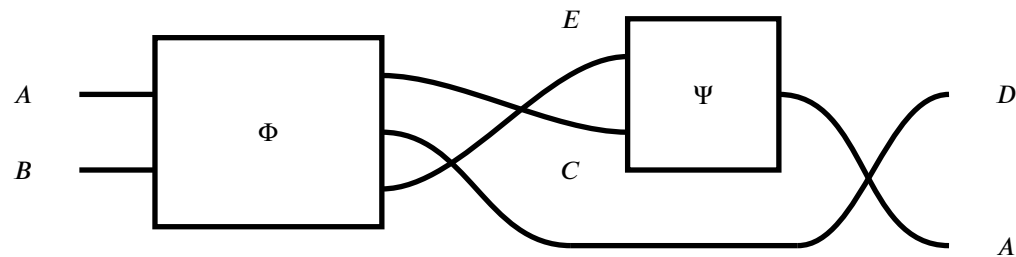
Processes may compose in parallel, which we depict as vertically stacking boxes.



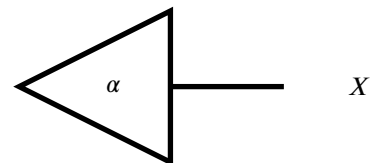
Processes may compose sequentially, which we depict as connecting wires of the same type from left to right.



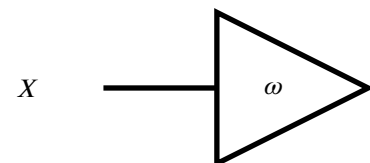
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



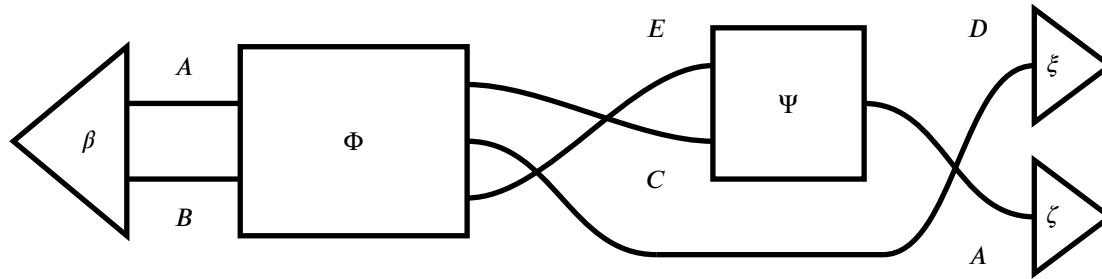
Some processes have no inputs; we call these *states*.



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 data¹:

- 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 $f : A \rightarrow B$. Sequential composition is function composition. Parallel composition of systems is the cartesian product of sets: the set of ordered pairs of two sets.

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

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 $f \otimes g : A \times C \rightarrow B \times D$ of functions $f : A \rightarrow B$ and $g : C \rightarrow D$ is defined:

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

The unit of parallel composition is the³ 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 $\Phi^* : A \rightharpoonup B$ or $\Phi_* : B \rightharpoonup A$.⁶ Sequential composition is relation composition:

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

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

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

The unit of parallel composition is the singleton. States 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 V , which we assume includes a choice of basis. The copy map is the rectangular matrix made of two identity matrices:

$$\Delta_V : V \rightarrow V \oplus V := \begin{bmatrix} \mathbf{1}_V & \mathbf{1}_V \end{bmatrix}$$

The delete map is the column vector of 1s:

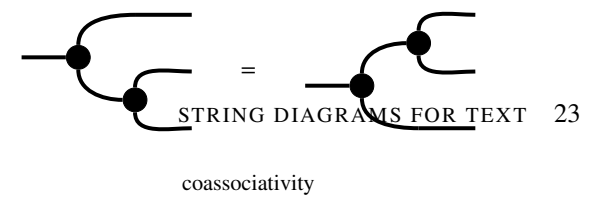
$$\epsilon_V : V \rightarrow \mathbf{0} := \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

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:

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

The delete function is defined:

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

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

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

The delete relation is defined:

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

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:

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.

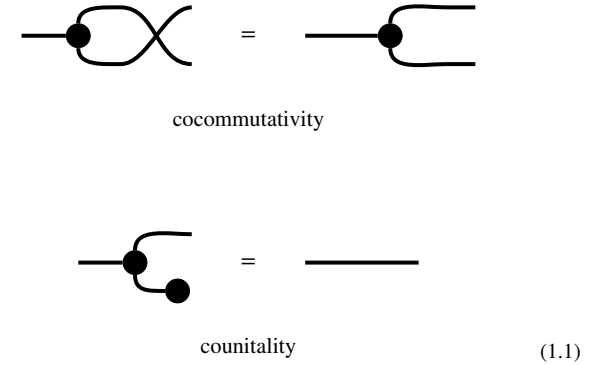
Coassociativity: says there is no difference between copying copies.

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

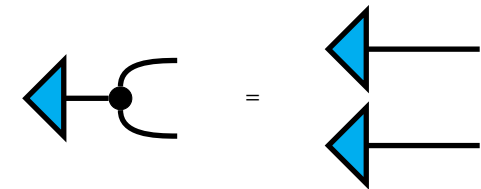
Counitality: says that if a copy is made and one of the copies is deleted, the remaining copy is the same as the original.

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

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 sub-theory 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 $\top : \{\star\} \rightharpoonup B := \{(\star, 0), (\star, 1)\} \simeq \{0, 1\}$. 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.

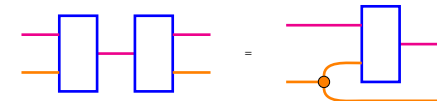
PutPut: Putting in one value and then a second is the same as deleting the first value and just putting in the second.



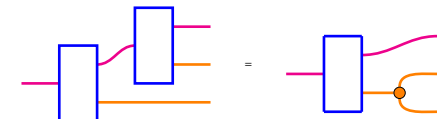
GetPut: Getting a value from a field and putting it back in is the same as not doing anything.

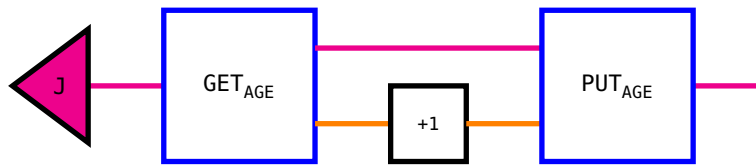


PutGet: Putting in a value and getting a value from the field is the same as first copying the value, putting in one copy and keeping the second.



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

1.4 What did Montague think "grammar" was?

WHAT IS THIS? The aim is to help the modern mathematician, familiar with some notions of Category Theory, in digesting Richard Merritt Montague's seminal 1970 paper *Universal Grammar*.

WHY MIGHT I CARE? Montague's introduction of formal methods to semantics revolutionised formal semantics, mirroring the closely preceding Chomskyian revolution in syntax of the 1960s [].

WHAT ARE YOUR MOTIVES? One might find it surprising that despite the breadth of Montague's influence today, it is quite hard to find a formal definition of what Montague semantics *is*. The answers naturally depend on what one considers important, but one answer is the literalist "what Montague himself said." He was, after all, a logician.

SO WHY NOT JUST READ *Universal Grammar*?

Montague himself considered part of his work a formal reification of the Fregean programme of compositionality, as evidenced by a culminating definition of "Fregean interpretations" in *Universal Grammar*. Therein lies the problem: he was a logician writing about structure preservation and compositionality in 1970, before the field of Category Theory had become the only game in town for formal accounts of these things. Reading the definitions feels like debugging punchcards.

IT'S BEEN 50 YEARS, WHAT ELSE IS THERE TO BE SAID?

There has been *a lot* of pollination from mathematics to linguistics since Chomsky and Montague, and formal semantics has largely matured past the kind of compositionality that Montague envisioned [formsem]. So keeping in mind that this is a historical exercise, I'll begin saying what can be said.

1.4.1 Historical Context and Scope of Analysis

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 were not *mathematically* novel – the lambda calculus had been around since [DATE], and Tarski and Carnap had been developing intensional higher-order logics since [] – but for linguists who, by-and-large, only knew first order predicate logic, these methods were a tour-de-force that solved longstanding problems in formal semantics.

Thus, Montague semantics has largely been in the care of linguists rather than mathematicians. This meant sparse opportunity for the ideas to 'update' according to mainstream developments in mathematics.

⁸

⁹

HOW MUCH OF IT IS WORTH TRANSLATING?

There is a natural division of Montague's approach into two structural components: the notion of a structure-preserving map from Syntax to Semantics, and the use of a powerful logic. The first may appeal more to philosophically-minded Fregeans and modern category theorists. The second

According to Partee [], a formal semanticist, advocate, and torch-bearer for Montague, the chief interest of Montague's approach (as far as his contemporary *linguists* were concerned) lay in the following ideas:

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

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

1.4.2 On "Background Notions"

Montague's *algebras* are clones, which are in bijection with Lawvere Theories

In Section 1, 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 1.4.1 (Generating data of an Algebra). Let A be the carrier set, and F_γ be a set of functions $A^k \rightarrow A$ for some $k \in N$, 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:

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

[[[maybe montague didn't want to entertain a nullary operation: why?]]]

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:

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

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.

Definition 1.4.2 is equivalent to asking for all projections. Definitions 1.4.2 and 1.4.4 together characterise Montaguevian 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 (\dagger) some operations F go untyped, or that (\ddagger) some are typed multiply.

Taking a disambiguated language \mathcal{U} on a carrier set A , Montague defines a *language* to be a pair $L := \langle \mathcal{U}, R \rangle$, where R is a relation from a subset of the carrier A to a set PE_L , the set of *proper expressions* of the language L . An admirable purpose of R appears to be to permit the modelling of *syntactic ambiguity*, where multiple elements of the term algebra \mathcal{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 (\dagger) obliquely, by defining ME_L to be the image in PE_L of R of just those expressions among A that are typed. Nothing appears to guard against (\ddagger), 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 consequence, in conjunction with (\star), is that every multiply typed operation F induces a boolean algebra where the typings are the generators and the operations are elementwise in the inputs and output. Worse problems occur, as Montague’s clone definition include all projectors, and when defined separately from the typing structure, these projectors may be typed in a way that permits operations that arbitrarily change types, which appears to defeat the purpose. We doubt these artefacts are intentional, so we will interpret Montague assuming his intent was a type-system 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 $(\Delta, \delta_0 : 1 \rightarrow \Delta)$.

2

Text circuits

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. typological 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 typological 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:

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

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:

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

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) $\forall x \exists y : \text{loves}(x, y)$. The odd reading is $\exists y \forall x : \text{loves}(x, y)$: a Raymond situation where there is a single person loved by everyone. We can sketch this difference formally using a simple combinatory categorial grammar.

$$\frac{\frac{\frac{\text{everyone} : (n \multimap s) \multimap s}{\text{everyone_loves} : s \multimap n} \quad \frac{\text{loves} : n \multimap s \multimap n}{\text{someone} : (s \multimap n) \multimap s}}{\text{everyone_loves_someone} : s}}$$

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

Bob drinks **God knows** how many beers

Bob drinks **who knows** how many beers

But it is awkward to have:

Bob drank **nobody knows** how many beers

And it is not acceptable to have:

Bob drank **Alice knows** how many beers

$$\frac{\text{everyone} : (n \multimap s) \multimap s \quad \frac{\text{loves} : n \multimap s \multimap n \quad \text{someone} : (s \multimap n) \multimap s}{\text{loves_someone} : n \multimap s}}{\text{everyone_loves_someone} : s}$$

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:

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

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

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

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.

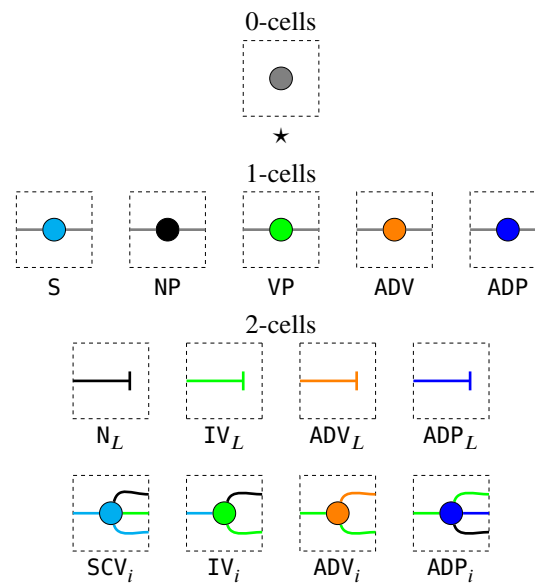
$$\frac{\frac{\lambda(\lambda x.V(x)).\ulcorner \forall x : V(x) \urcorner : (n \multimap s) \multimap s \quad \lambda x \lambda y.\ulcorner \text{loves}(x, y) \urcorner : n \multimap s \multimap n}{\lambda y.\ulcorner \forall x : \text{loves}(x, y) \urcorner : s \multimap n} \quad \lambda(\lambda y.V(y)).\ulcorner \exists y : V(y) \urcorner : (s \multimap n) \multimap s}{\ulcorner \exists y \forall x : \text{loves}(x, y) \urcorner : s}$$

$$\frac{\lambda(\lambda x.V(x)).\ulcorner \forall x : V(x) \urcorner : (n \multimap s) \multimap s \quad \frac{\lambda x \lambda y.\ulcorner \text{loves}(x, y) \urcorner : n \multimap s \multimap n \quad \lambda(\lambda y.V(y)).\ulcorner \exists y : V(y) \urcorner : (s \multimap n) \multimap s}{\lambda x.\ulcorner \exists y : \text{loves}(x, y) \urcorner : n \multimap s}}{\ulcorner \forall x \exists y : \text{loves}(x, y) \urcorner : s}$$

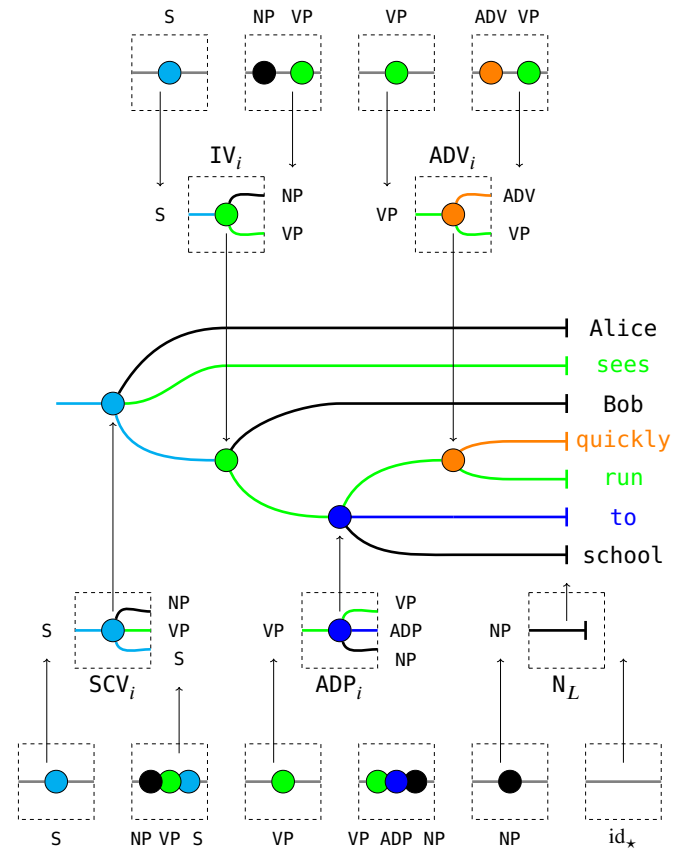
THE SPEAKER'S CHOICES AND THE LISTENER'S DEDUCTIONS MUST BE RELATED. Even if the pair are cooperating, so that the speaker tries to avoid ambiguity,

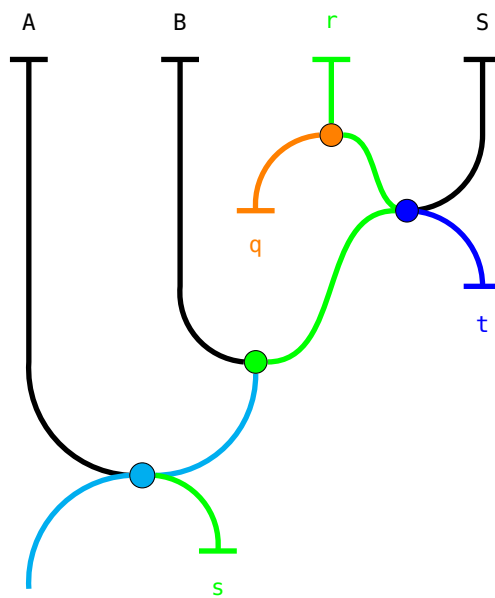
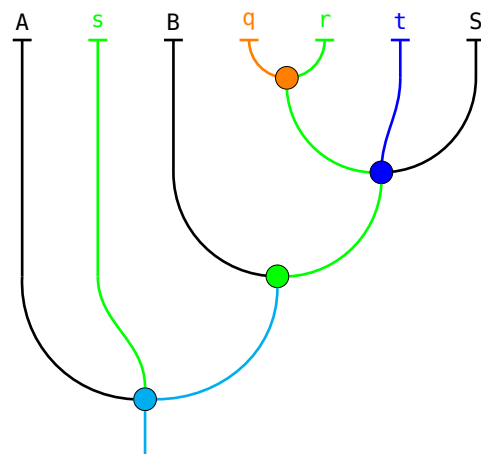
2.1.2 A context free grammar to generate *Alice sees Bob quickly run to school*

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.



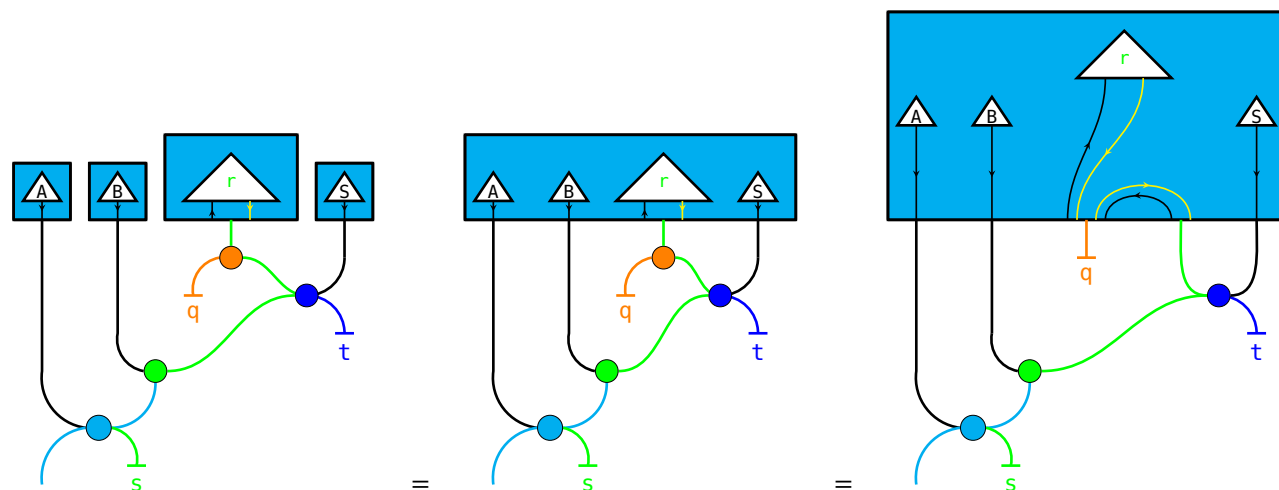
Consider the sentence *Alice sees Bob quickly run to school*, which we take to be generated by the following context-free grammar derivation, read from left-to-right. We additionally depict the breakdown of the derivation in terms of rewrites of lower dimension from our signature.



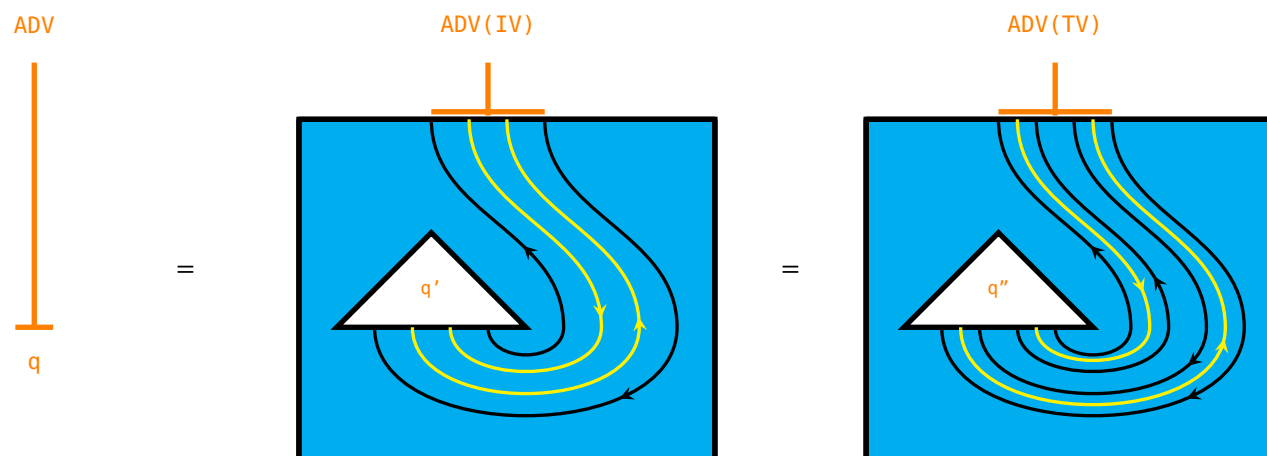


2.1.3 Relating the generative grammar to a pregroup grammar via a discrete monoidal fibration

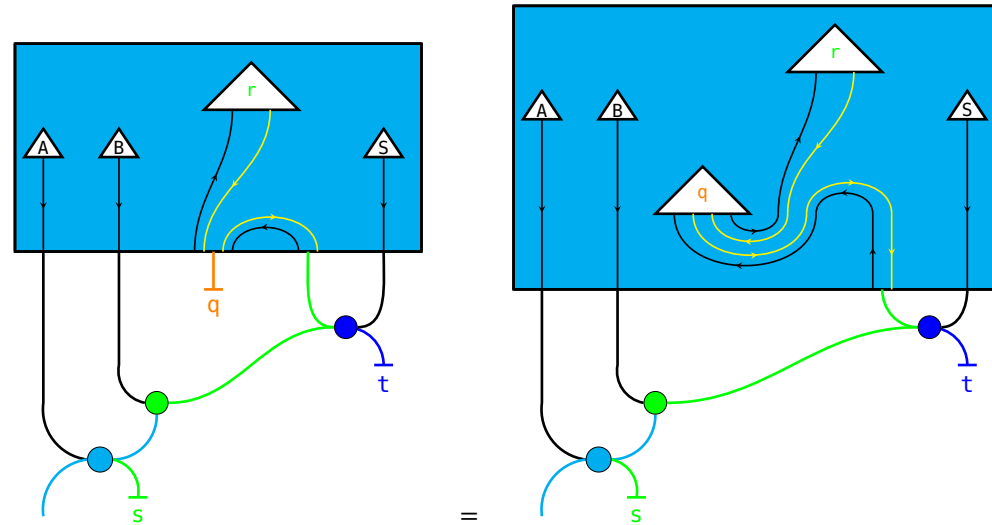
We merge the monoidal functor-boxes and we slide the bottom edge down using the fibration.



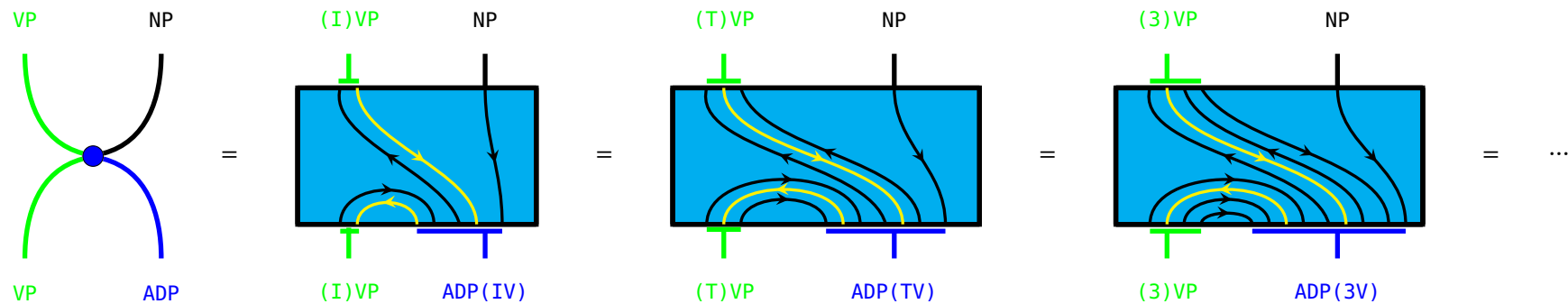
quickly could find itself modifying an intransitive (single noun) or transitive (two noun) verb. Suppose that it is the job of some process q' to handle intransitive verbs, and similarly q'' to handle transitive ones. We use the functor for bookkeeping, by asking it to send both q' and q'' to the dependent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



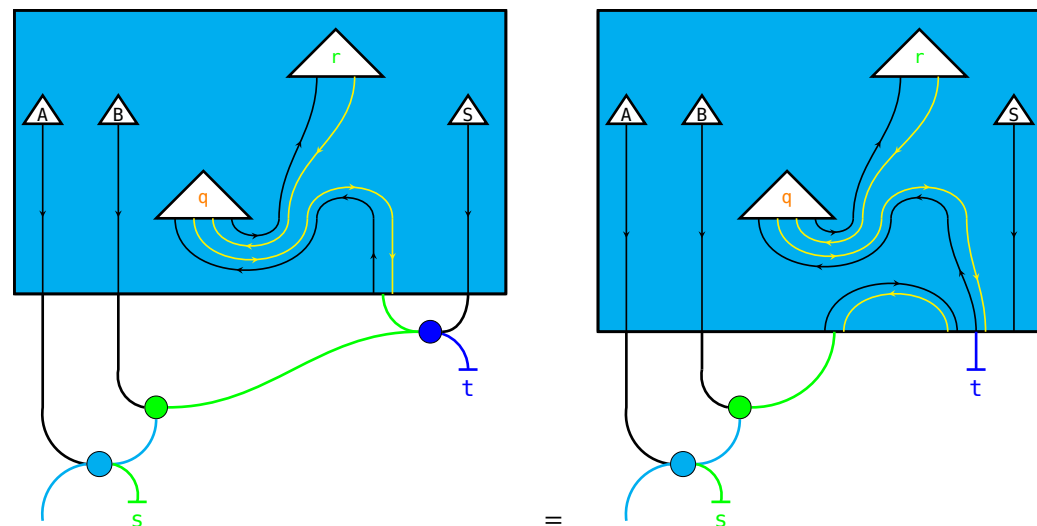
Since the functor is a monoidal discrete fibration, it introduces the appropriate choice of quickly when we pull the functor-box down, while leaving everything else in parallel alone.



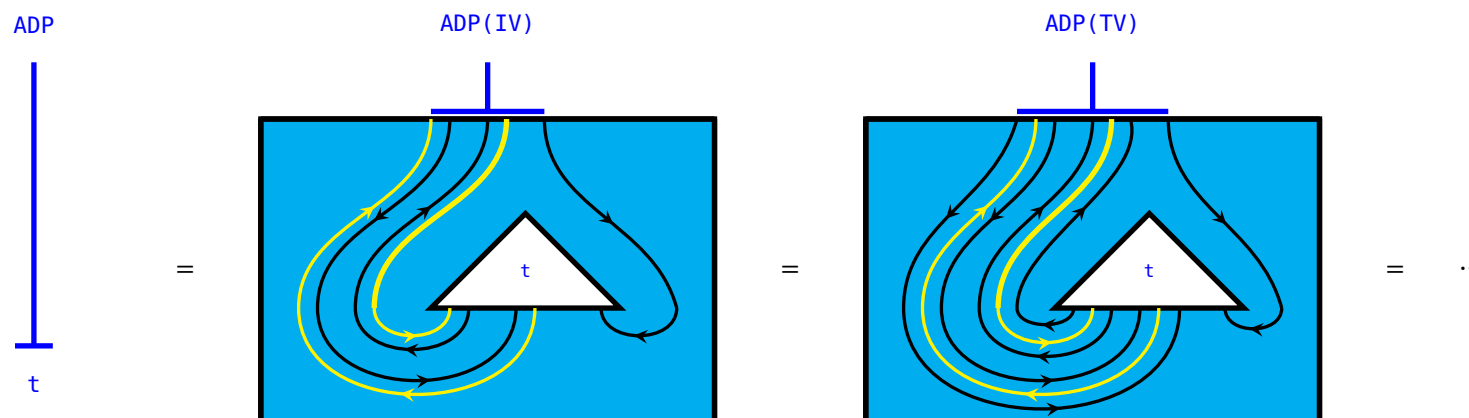
Adpositions can apply for verbs of any noun-arity. We again use the fiber of the functor for bookkeeping by asking it to send all of the following partial pregroup diagrams to the adposition generator. We consider the pregroup typing of a verb of noun-arity $k \geq 1$ to be ${}^{-1}n \cdot s \cdot \underbrace{n^{-1}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



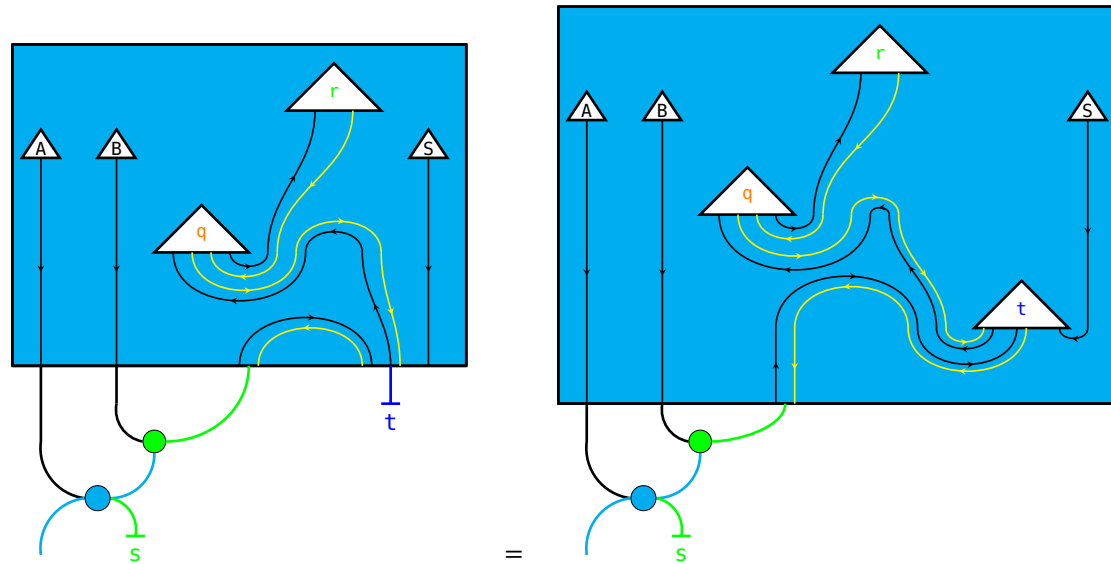
When we pull down the functor-box, the discrete fibration introduces the appropriate choice of diagram from above, corresponding to the intransitive verb case.



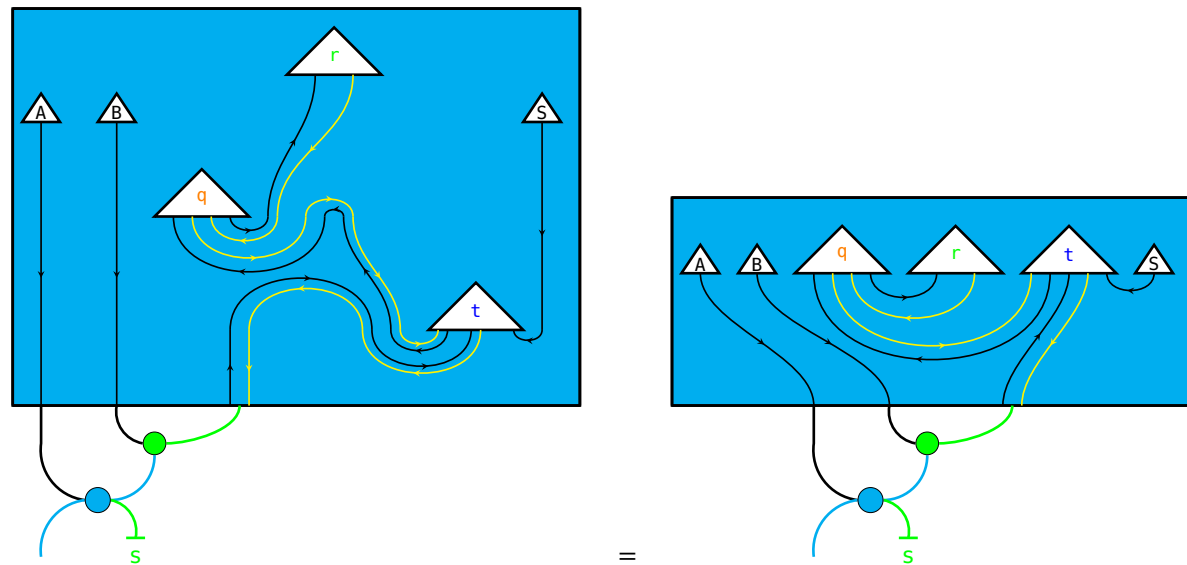
Similarly to quickly, we suppose we have a family of processes for the word to, one for each noun-arity of verb.



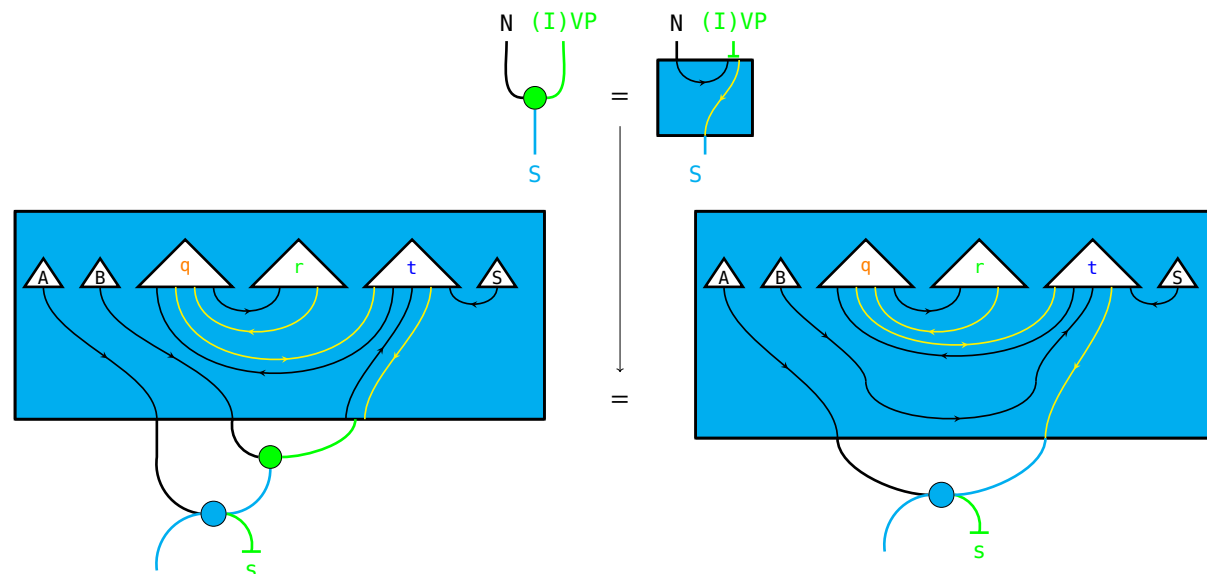
Again the discrete fibration introduces the appropriate choice of t when we pull the functor box down.



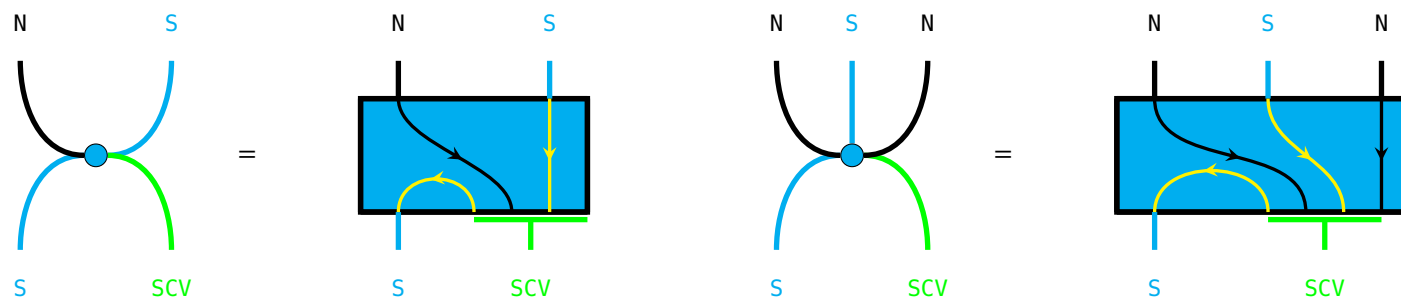
Now we visually simplify the inside of the functor-box by applying yanking equations.



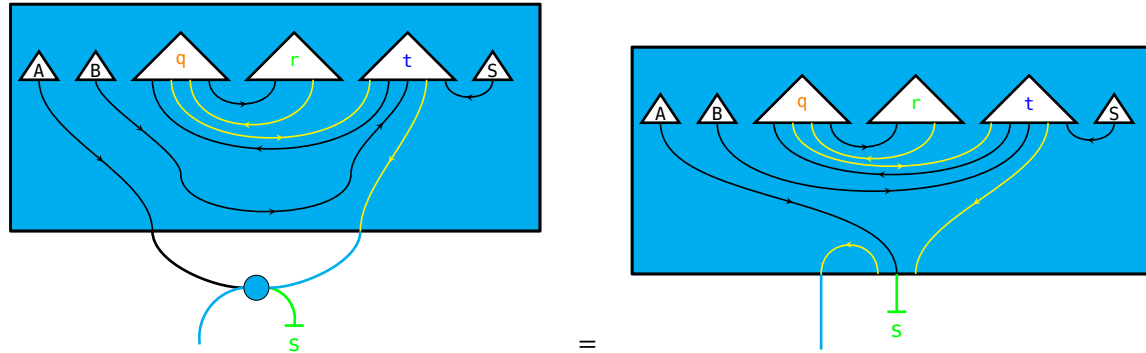
Similarly as before, we can pull the functor-box past the intransitive verb node. There is only one pregroup type $^{-1}n \cdot s$ that corresponds to the grammatical category $(I)VP$.



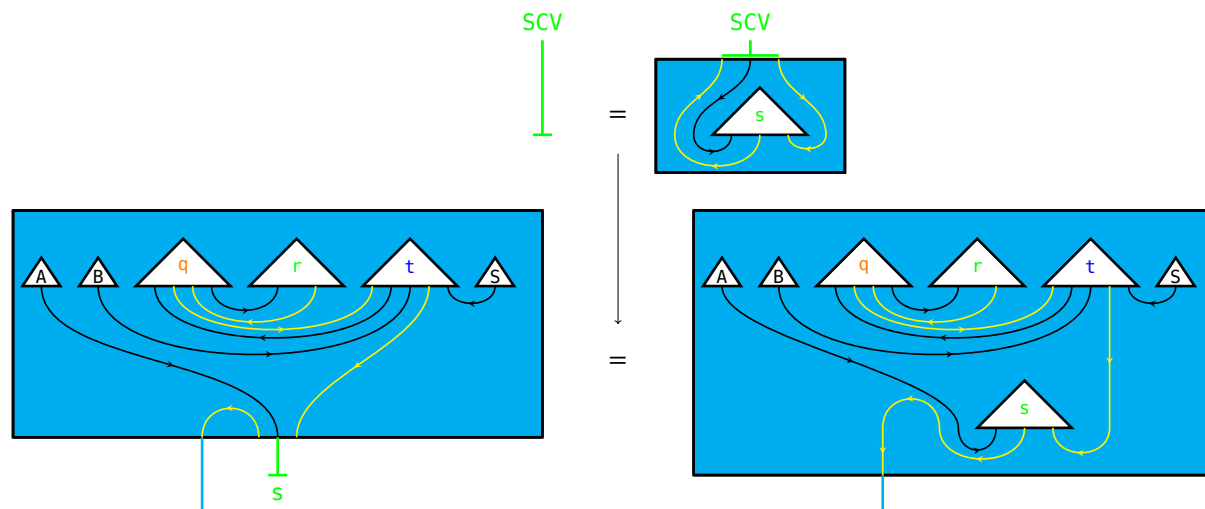
Proceeding similarly, we can pull the functor-box past the sentential-complement-verb node. There are multiple possible pregroup types for SCV , depending on how many noun-phrases are taken as arguments in addition to the sentence. For example, in Alice *sees* [sentence], *sees* returns a sentence after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



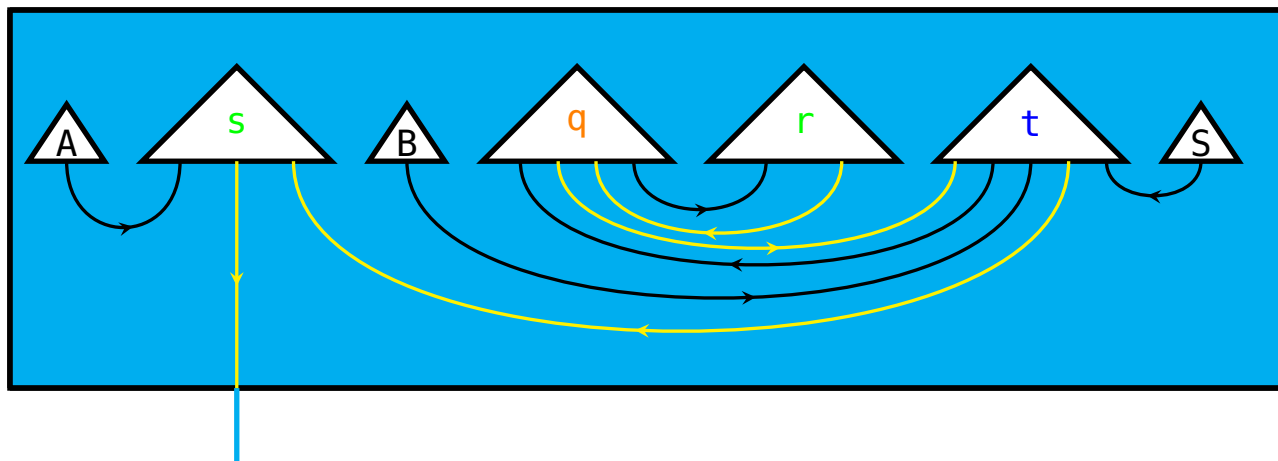
Pulling down the functor box:



As before, we can ask the functor to send an appropriate partial pregroup diagram to the dependent label $\bar{s}ee$.



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.

$$\begin{array}{c}
 \frac{q : (-^1 n \cdot s) \cdot (-^1 n \cdot s)^{-1} \quad r : -^1 n \cdot s}{q _ r : -^1 n \cdot s} \quad \frac{t : -^1 (-^1 n \cdot s) \cdot (-^1 n \cdot s) \cdot n^{-1}}{q _ r _ t : (-^1 n \cdot s) \cdot n^{-1}} \quad S : n \\
 \frac{A : n \quad s : -^1 n \cdot s \cdot s^{-1}}{A _ s : s \cdot s^{-1}} \quad B : n \quad \frac{B _ q _ r _ t _ S : s}{A _ s _ B _ q _ r _ t _ S : s}
 \end{array}$$

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.

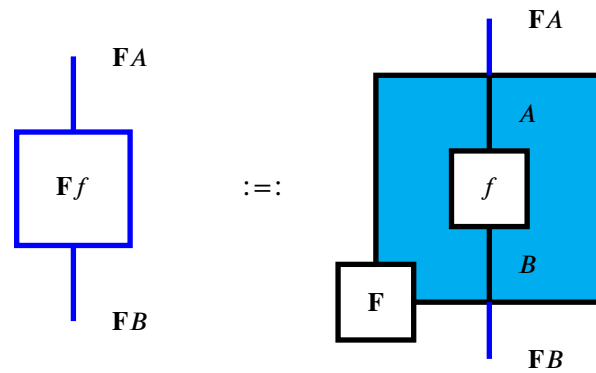
To enforce the well-definedness of the functor $\mathbf{F} : \mathcal{PG} \rightarrow \mathcal{G}$ on objects, we may consider the strictified "category of..." [ghicadiagrams]...

2.1.4 Discrete Monoidal Fibrations

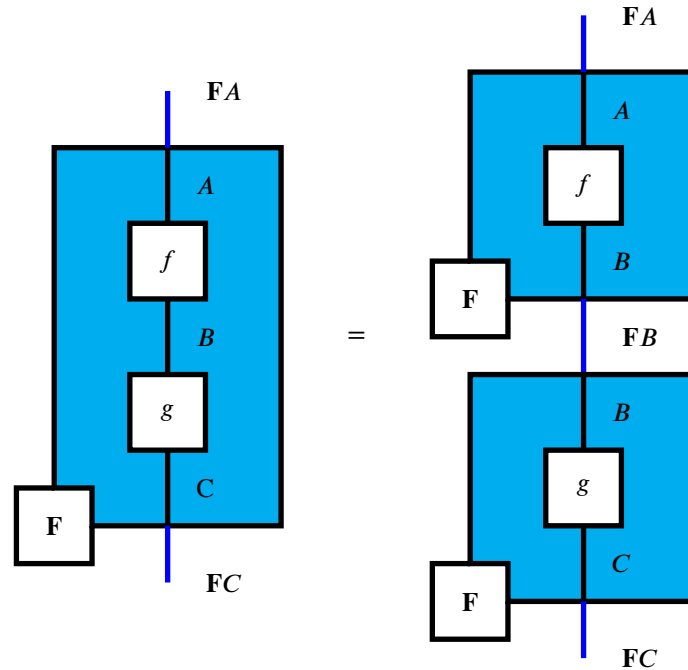
We introduce the concept of a discrete monoidal fibration: a mathematical bookkeeping tool that relates kinds of choices speakers and listerns make when generating and parsing text respectively. We proceed diagrammatically. The first concept is that of a *monoidal functor box*.

Suppose we have a functor between monoidal categories $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$. Then we have the following diagrammatic representation of a morphism $\mathbf{F}A \mapsto \mathbf{F}B$ 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 [monoidal/fibration]. 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].

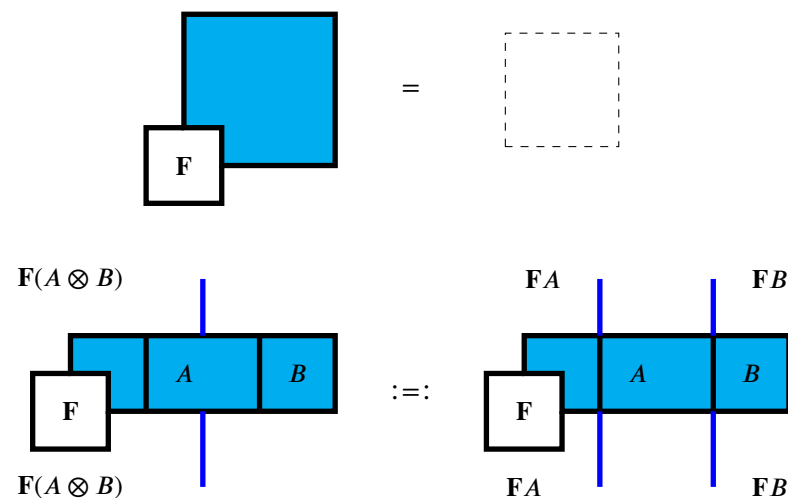


The use of a functor box is like a window from the target category D into the source category C ; when we know that a morphism in D is the image under F of some morphism in C , the functor box notation is just a way of presenting all of that data at once. Since F is a functor, we must have that $Ff;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

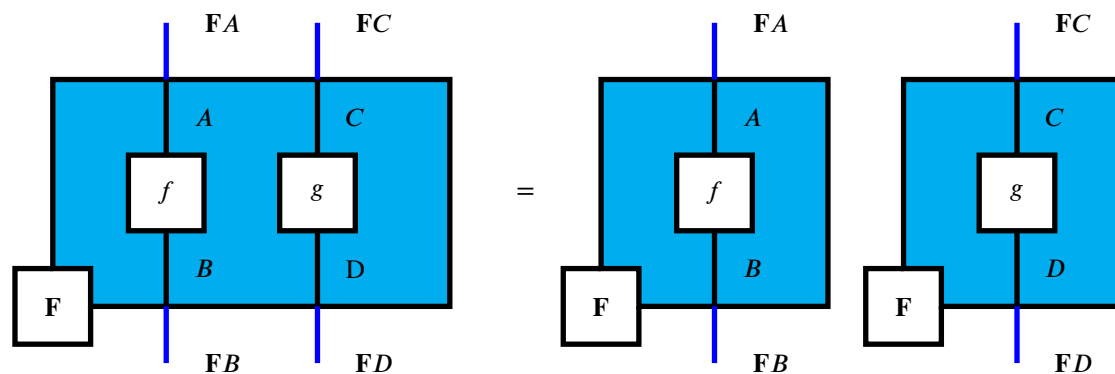


Assume that F is strict monoidal; without loss of generality by the strictification theorem [], this lets us gloss over the associators and unitors. For F to be strict monoidal, it has

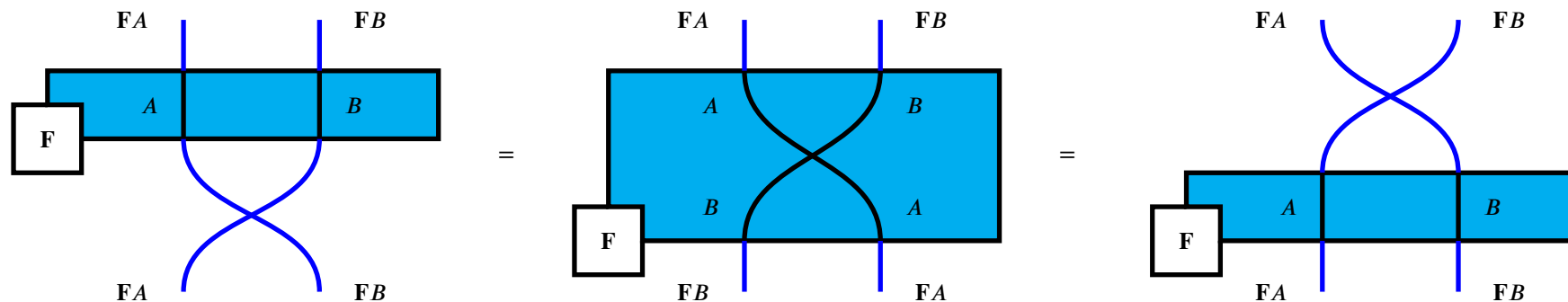
to preserve monoidal units and tensor products on the nose: i.e. $\mathbf{F}I_C = I_D$ and $\mathbf{F}A \otimes_D \mathbf{F}B = \mathbf{F}(A \otimes_C B)$. Diagrammatically these structural constraints amount to the following equations:



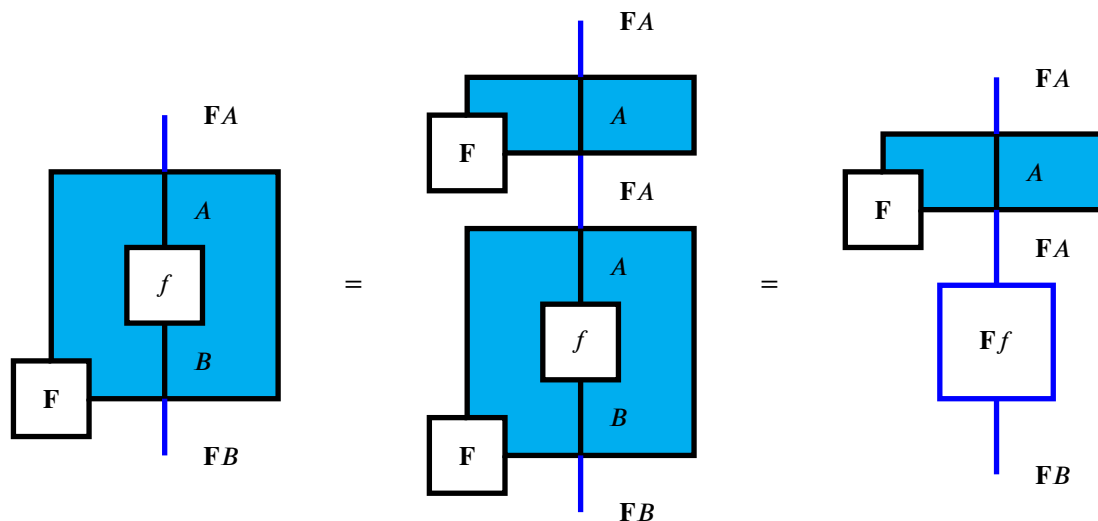
What remains is the monoidality of \mathbf{F} , which is the requirement $\mathbf{F}f \otimes \mathbf{F}g = \mathbf{F}(f \otimes g)$. Diagrammatically, this equation is represented by freely splitting and merging functor boxes horizontally; analogously to how splitting vertically is the functor-boxes'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

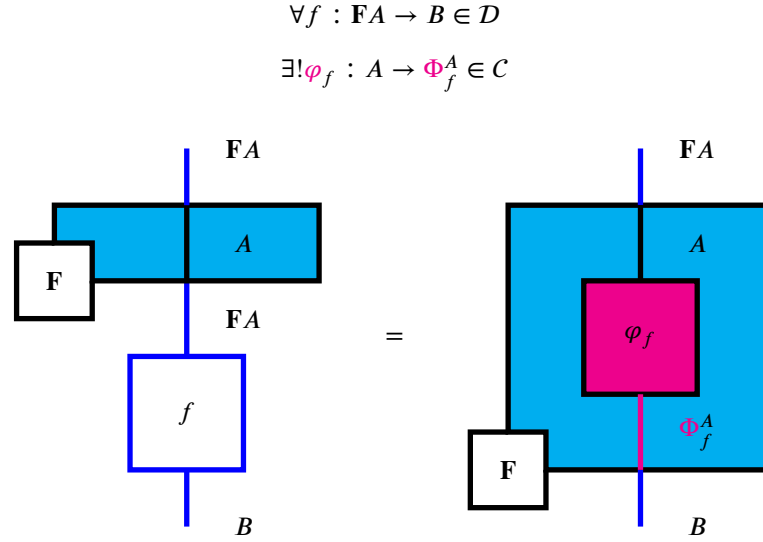


When can we do the reverse? That is, take a morphism in \mathcal{D} and *slide it into* a functor box? We know that in general this is not possible, because not all morphisms in \mathcal{D} may be in the image of \mathbf{F} . So instead we ask "under what circumstances" can we do this for a functor \mathbf{F}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

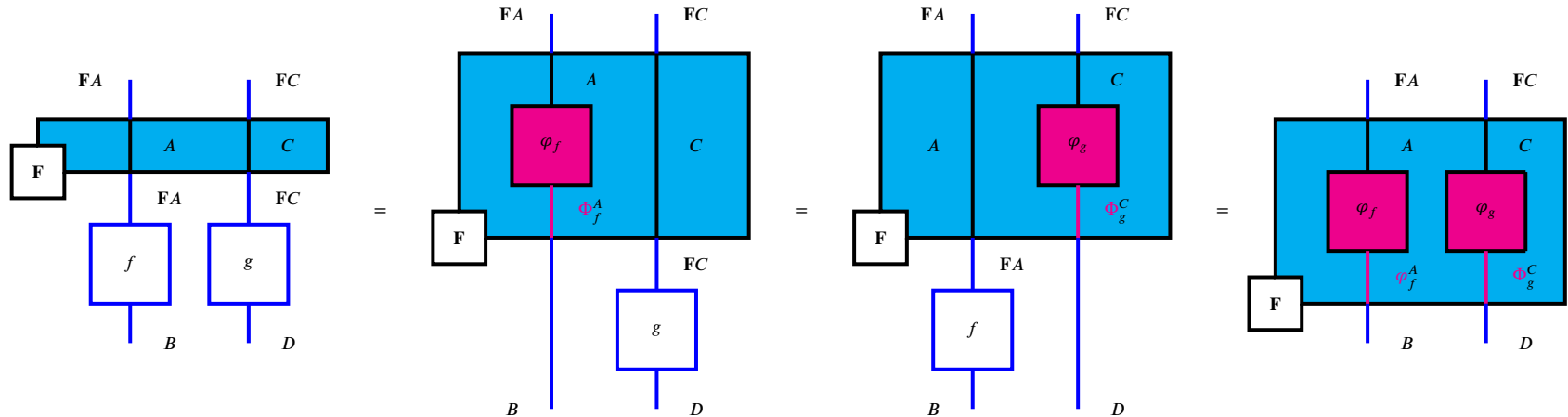
Definition 2.1.7 (Discrete opfibration). $\mathbf{F} : \mathcal{C} \rightarrow \mathcal{D}$ is a *discrete fibration* when:

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

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



Definition 2.1.8 (Monoidal discrete opfibration). We consider \mathbf{F} to be a (*strict, symmetric*) *monoidal discrete opfibration* when it is a (strict, symmetric) monoidal functor, a discrete opfibration, and the 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 definition 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 $(\mathcal{N}, \mathcal{N}^\downarrow, \mathcal{N}^\star, \Sigma, \mathcal{I}, \mathcal{A}, s \in \mathcal{N})$. These denote, respectively:

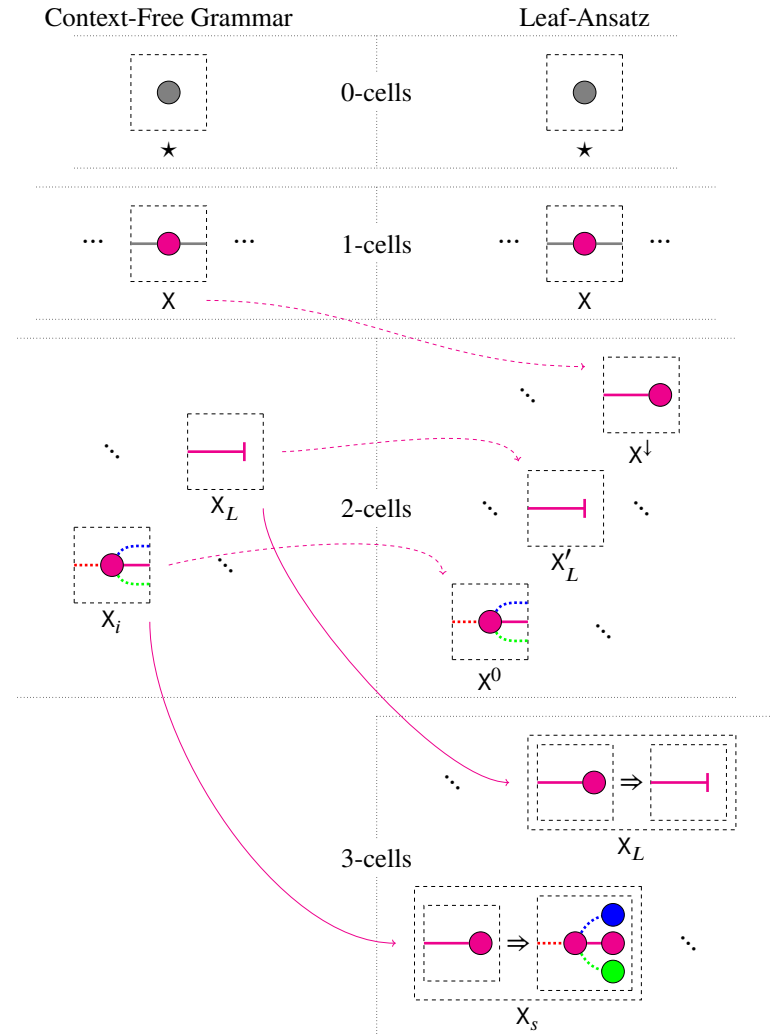
- The *non-terminals*:
 - A set of *non-terminal symbols* \mathcal{N} – these stand in for grammatical types such as NP and VP.
 - A bijection $\downarrow: \mathcal{N} \rightarrow \mathcal{N}^\downarrow$ which acts as $X \mapsto X^\downarrow$. Nonterminals in \mathcal{N} are sent to marked counterparts in \mathcal{N}^\downarrow , and the inverse sends marked nonterminals to their unmarked counterparts. These markings are *substitution markers*, which are used to indicate when certain leaf nodes are valid targets for a substitution operation – discussed later.
 - A bijection $\star: \mathcal{N} \rightarrow \mathcal{N}^\star$ – 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 $X^\star \in \mathcal{N}^\star$; 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 X^\downarrow in a tree α with another tree α' that has X as a root node.
- *Adjoining* takes auxiliary tree β with root and foot nodes X, X^\star , and a derived tree γ at an interior node X of γ . Removing the X node from γ separates it into a parent tree with an X-shaped hole for one of its leaves, and possibly multiple child trees with X-shaped holes for roots. The result of adjoining is obtained by identifying the root of β with the X-context of the parent, and making all the child trees children of β 's foot node X^\star .

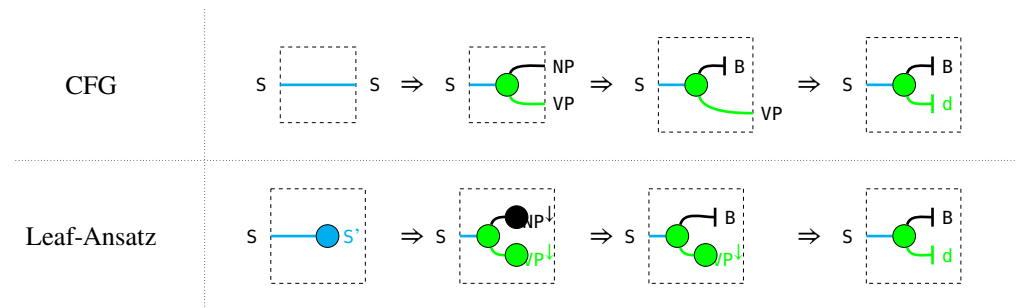
The essence of a *tree-adjoining grammar* is as follows: whereas for a CFG one grows the tree by appending branches and leaves at the top of the tree (substitution), in a TAG one can also sprout subtrees from the middle of a branch (adjoining). Now we show that this gloss is more formal than it sounds, 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 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}' .



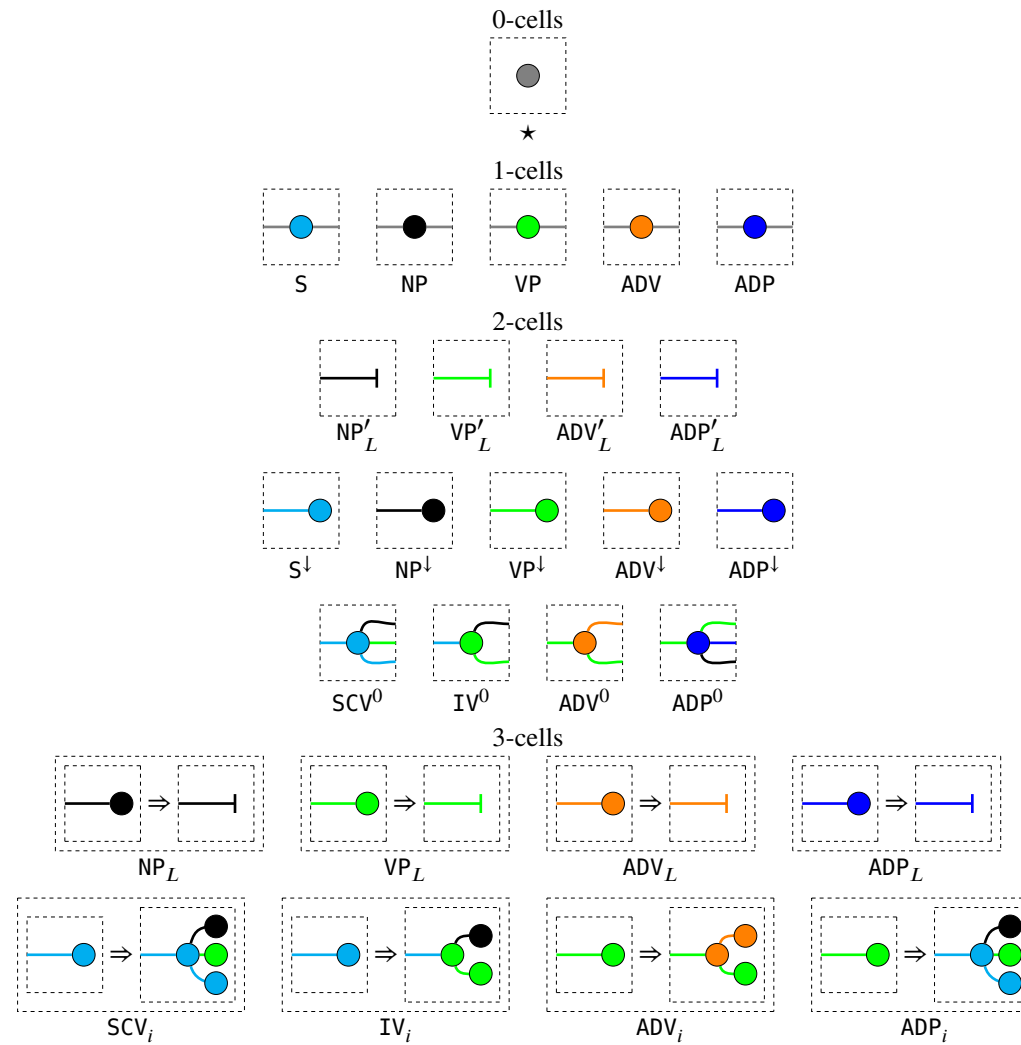
Example 2.1.12. The leaf-ansatz construction just makes formal the following observation: there are multiple equivalent ways of modelling terminal symbols in a rewrite system considered string-diagrammatically. One way (which we have already done) is to treat non-terminals as wires and terminals as effects,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



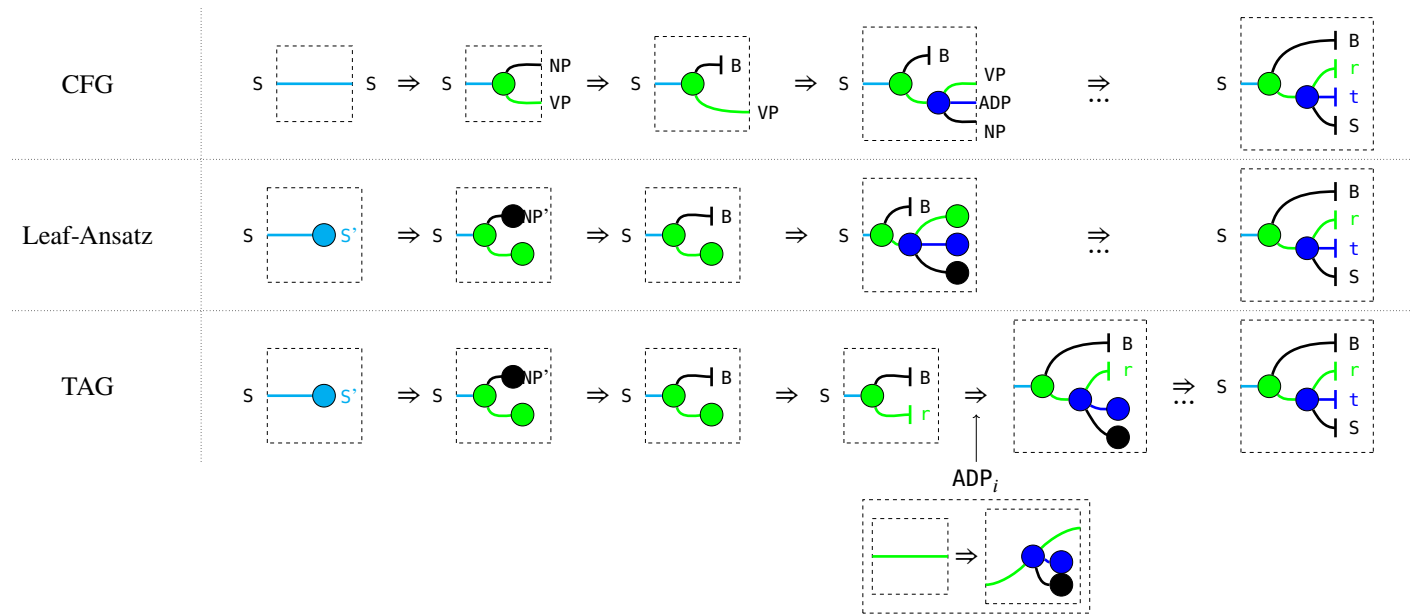
Proposition 2.1.13 (Leaf-ansatzes of CFGs are precisely TAGs with only initial trees and substitution). *Proof.* By construction. Consider a CFG given by 2-categorical signature \mathfrak{G} , with leaf-ansatz signature \mathfrak{G}' . The types X of \mathfrak{G} become substitution marked symbols X^\downarrow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

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



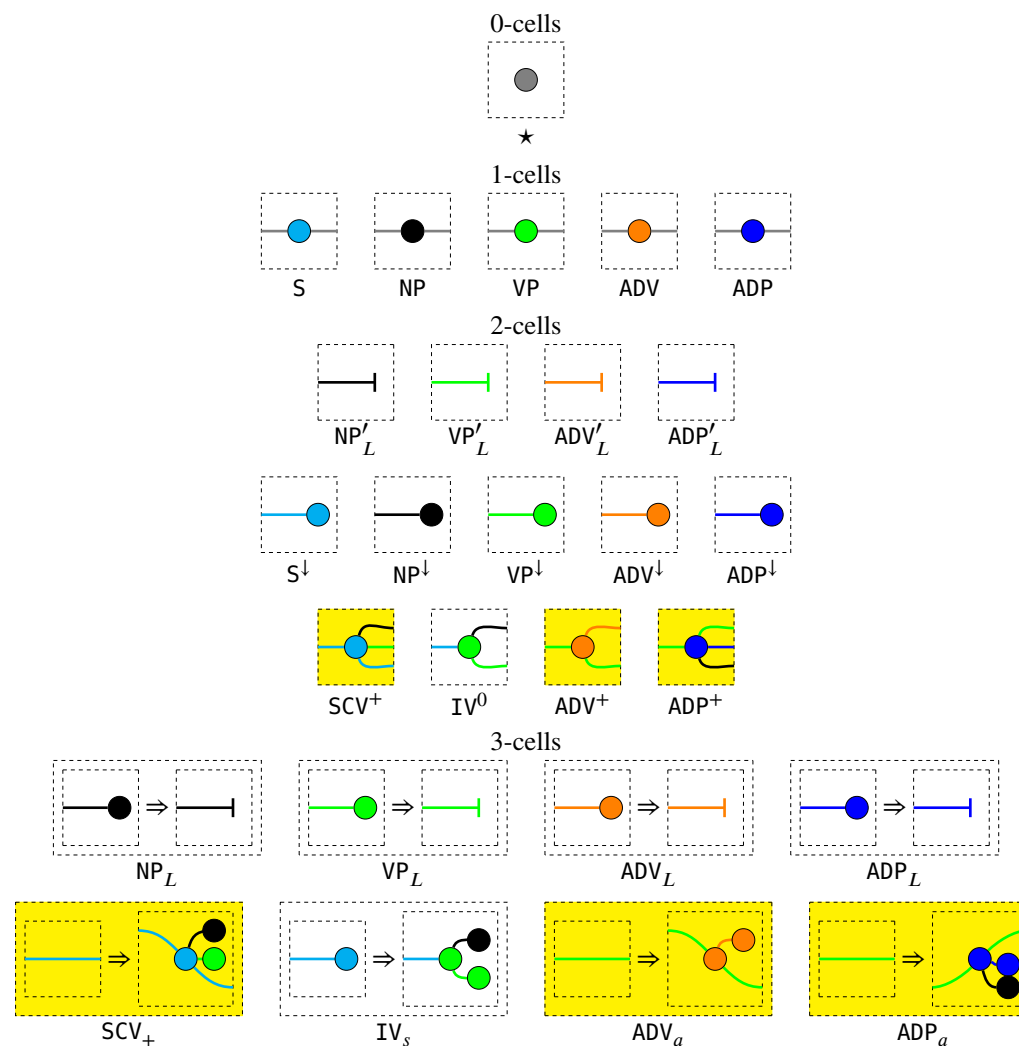
Example 2.1.15 (Adjoining is sprouting subtrees in the middle of branches). One way we might obtain the sentence Bob runs to school is to start from the simpler sentence Bob runs, and then refine the verb runs into runs to school. This refinement on part of an already completed sentence is not permitted 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.

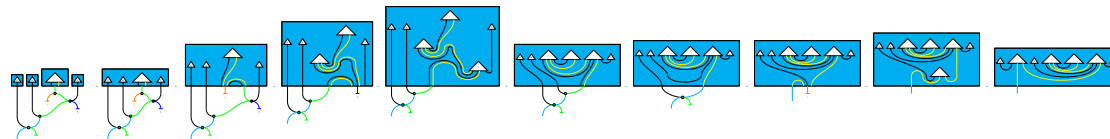


The construction yields as a corollary an alternate proof of Theorem [Joshi 6.1.1...].

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

Proof. Proposition 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 X, X^* indicate which wire-type to take as the identity in the left of the 3-cell, and the right of the 3-cell is obtained by replacing all non- X open wires Y with their leaf-ansatzes Y^\downarrow . This establishes a correspondence between any 2-cells of \mathfrak{G} considered as auxiliary trees in \mathfrak{G}' . \square



3

Continuous relations

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 $R : (X, \tau) \rightarrow (Y, \sigma)$ is a relation $R : X \rightarrow Y$ such that

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

where \dagger denotes the relational converse.

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

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

Arbitrary unions of opens are open

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

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

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

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

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

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

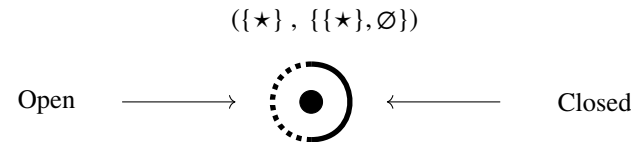
$$U \in \sigma \Rightarrow f^{-1}(U) \in \tau$$

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

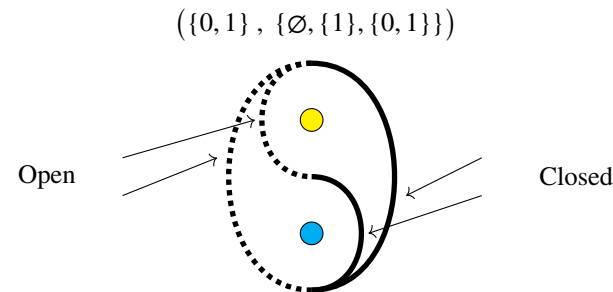
3.2 Continuous Relations by examples

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.

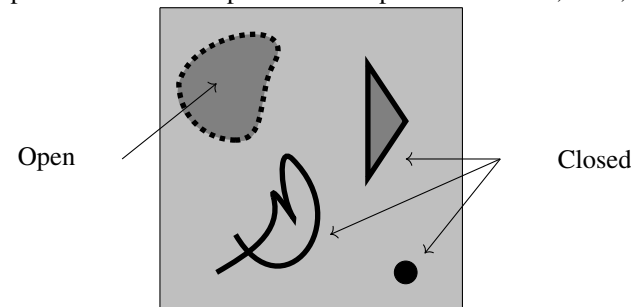
The **singleton space** consists of a single point which is both open and closed. We denote this space \bullet . Concretely, the underlying set and topology is



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 S . Concretely, the underlying set and topology is:



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



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

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

$S \rightarrow \bullet$: 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.

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

$\bullet \rightarrow \blacksquare$: Proposition 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.

$\blacksquare \rightarrow \bullet$: 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.

THERE ARE 16 CANDIDATE RELATIONS $S \rightarrow S$ 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 $X^\tau \rightarrow Y^\sigma$ ARE ALWAYS CONTINUOUS?

THE EMPTY RELATION IS ALWAYS CONTINUOUS.

Proposition 3.2.6. *Proof.* The preimage of the empty relation is always \emptyset , which is open by definition. \square

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. \square

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 \emptyset , which is open. If they overlap, the preimage of V is U , which is open. \square

CONTINUOUS FUNCTIONS ARE ALWAYS CONTINUOUS.

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

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

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

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

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

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

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

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

$$\emptyset \subset X \times Y$$

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

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

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.

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

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

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

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

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

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

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

A TOPOLOGICAL BASIS FOR SPACES OF CONTINUOUS RELATIONS

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

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

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

Lemma 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 $X^\tau \rightarrow Y^\tau$ that contains (x, y) .

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

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

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

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.

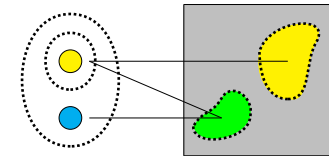


Figure 3.1: Regions of \blacksquare in the image of the yellow point alone will be coloured yellow, and regions in the image of both yellow and cyan will be coloured green:

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

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

Proof. We will show that every continuous relation $R : X^\tau \rightarrow Y^\sigma$ arises as a union of 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. \square

$S \rightarrow S$: 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.

$S \rightarrow \blacksquare$: Now we use the colour convention of the points in 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 $S \rightarrow \blacksquare$ in words as follows: Cyan only in points and lines, and either contained in or at the boundary of yellow or green. Have as much yellow and green as you like.

$\blacksquare \rightarrow S$: The preimage of all of 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.

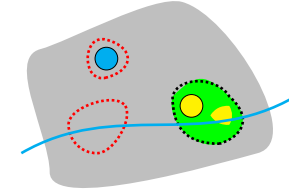


Figure 3.2: Regions in the image of the cyan point alone cannot be open sets by continuity, so they are either points or lines. Points and lines in cyan must be surrounded by an open region in either yellow or green, or else we violate continuity (open sets in red).

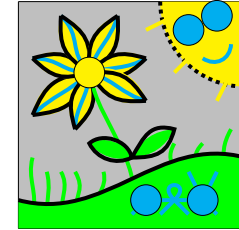


Figure 3.3: A continuous relation $S \rightarrow \blacksquare$: "Flower and critter in a sunny field".

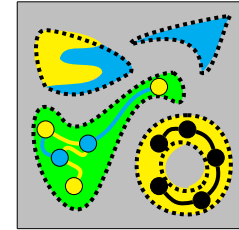


Figure 3.4: A continuous relation $\blacksquare \rightarrow S$: "still math?". Black lines and dots indicate gaps.

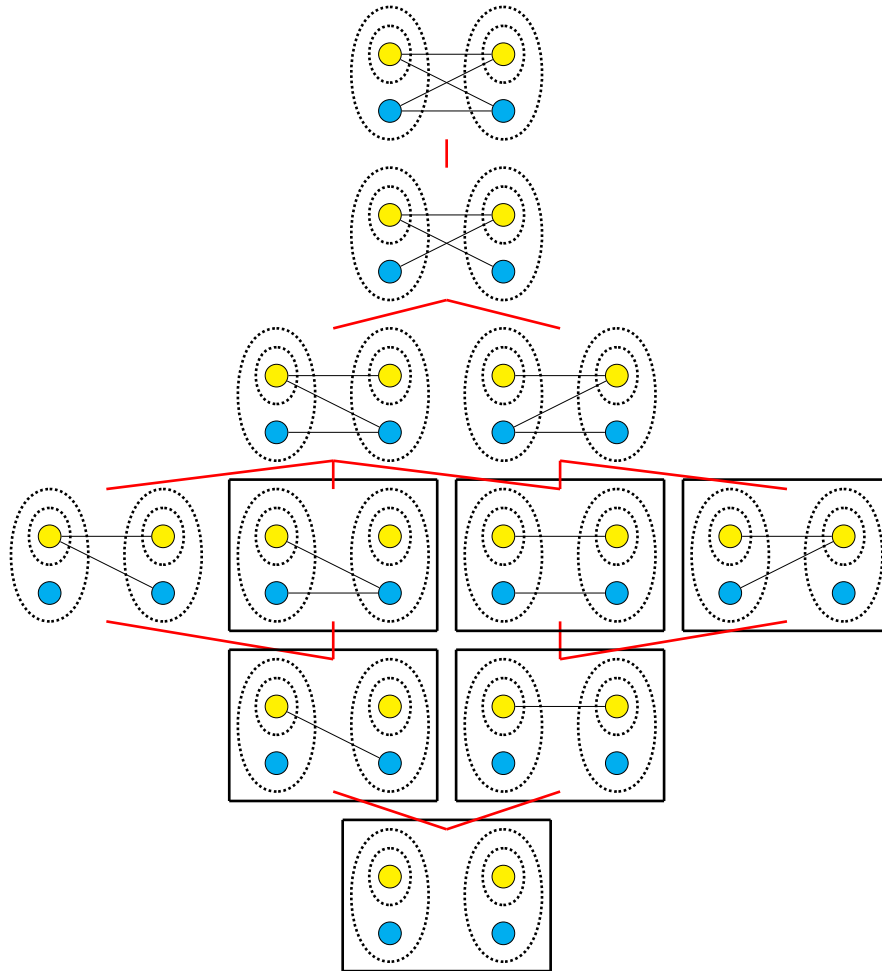


Figure 3.5: Hasse diagram of all continuous relations from the Sierpiński space to itself. Each relation is depicted left to right, and inclusion order is bottom-to-top. Relations that form the topological basis are boxed.

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

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

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



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



Figure 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."

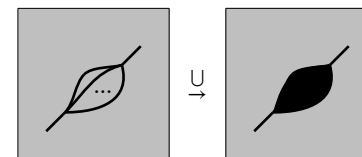


Figure 3.8: We can control the thickness of the brushstroke, by taking the union of (uncountably) many lines.



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 $[0, 1] \rightarrow \blacksquare$: "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 $X^\tau \xrightarrow{R} Y^\sigma \xrightarrow{S} Z^\theta$ of continuous relations is again continuous as follows:

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

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

3.4 Biproducts

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

Definition 3.4.1 ($F: \mathbf{Rel} \rightarrow \mathbf{TopRel}$). We define the action of the functor F :

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

On morphisms $F(X \xrightarrow{R} Y) := X^\star \xrightarrow{R} Y^\star$. 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 ($U: \mathbf{TopRel} \rightarrow \mathbf{Rel}$). We define the action of the functor U :

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

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

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.

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

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

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

Now we verify the triangle identities...

placeholder

□

Corollary 3.4.4. \mathbf{TopRel} has a zero object and biproducts.

Proof.

□

3.5 Symmetric Monoidal Closed structure

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

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

TENSOR PRODUCT: For objects, $X^\tau \otimes Y^\sigma$ has base set $X \times Y$ equipped with the product topology $\tau \times \sigma$. For morphisms, $R \otimes S$ the product of relations. We show that the tensor of continuous relations is again a continuous relation. Take continuous relations $R : X^\tau \rightarrow Y^\sigma$, $S : A^\alpha \rightarrow B^\beta$, and let U be open in the product topology $(\sigma \times \beta)$.

We need to prove that $(R \times S)^\dagger(U) \in (\tau \times \alpha)$. We may express U as $\bigcup_{i \in I} y_i \times b_i$, where the y_i and b_i are in the bases \mathfrak{b}_σ and \mathfrak{b}_β respectively. Since for any relations we have that $R(A \cup B) = R(A) \cup R(B)$ and $(R \times S)^\dagger = R^\dagger \times S^\dagger$:

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

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.

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

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

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

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

ASSOCIATORS:

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

BRAIDS:

...

COHERENCES:

...

□

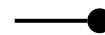
3.6 *TopRel* diagrammatically

3.6.1 Relations that are always continuous

HERE ARE FIVE CONTINUOUS RELATIONS FOR ANY X^τ :



everything



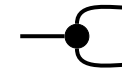
delete



nothing (state)

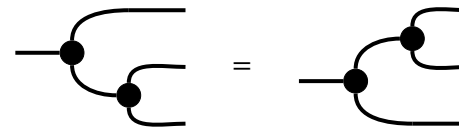


nothing (test)



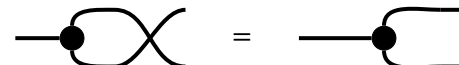
copy

COPY AND DELETE OBEY THE FOLLOWING EQUALITIES:



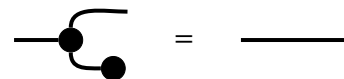
The diagram shows two equivalent ways to connect three horizontal lines. On the left, a line from the left enters a dot, which then splits into two lines that each enter another dot, which finally splits into two lines. On the right, a line from the left enters a dot, which splits into two lines; the top line enters a dot that splits into two lines, and the bottom line enters a dot that splits into two lines. An equals sign is between the two diagrams.

coassociativity



The diagram shows two equivalent ways to connect two horizontal lines. On the left, a line from the left enters a dot, which then splits into two lines that cross each other before entering another dot. On the right, a line from the left enters a dot, which splits into two lines that do not cross. An equals sign is between the two diagrams.

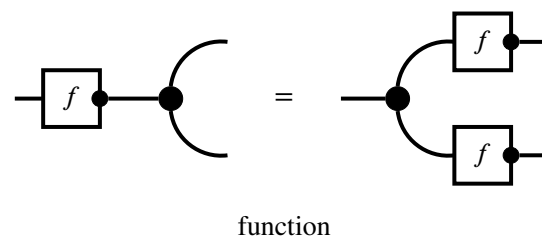
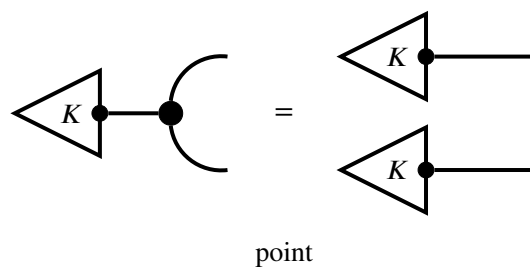
cocommutativity



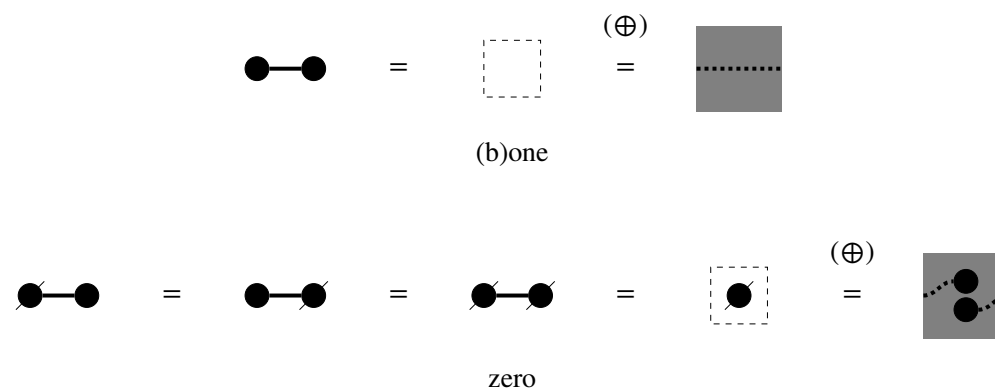
The diagram shows two equivalent ways to connect a single horizontal line. On the left, a line from the left enters a dot, which splits into two lines that each enter another dot, which finally splits into two lines. On the right, a single line continues straight. An equals sign is between the two diagrams.

counitality

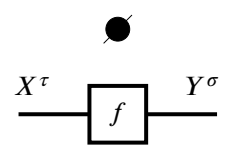
THE COPY MAP CAN ALSO BE USED TO DISTINGUISH THE DETERMINISTIC MAPS – POINTS AND FUNCTIONS – WHICH WE NOTATE WITH AN EXTRA DOT.



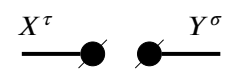
EVERYTHING, DELETE, NOTHING-STATES AND NOTHING-TESTS COMBINE TO GIVE TWO NUMBERS, ONE AND ZERO. There are extra expressions in grey squares above: they anticipate the tape-diagrams we will later use to graphically express another monoidal product of **TopRel**, the direct sum \oplus .



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



$\forall X^\tau \forall Y^\sigma \forall f$
 =
 zero



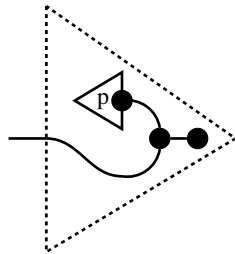
3.7 Populating space with shapes using sticky spiders

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 S with itself.

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

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

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

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

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

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

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

and *everything*:

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

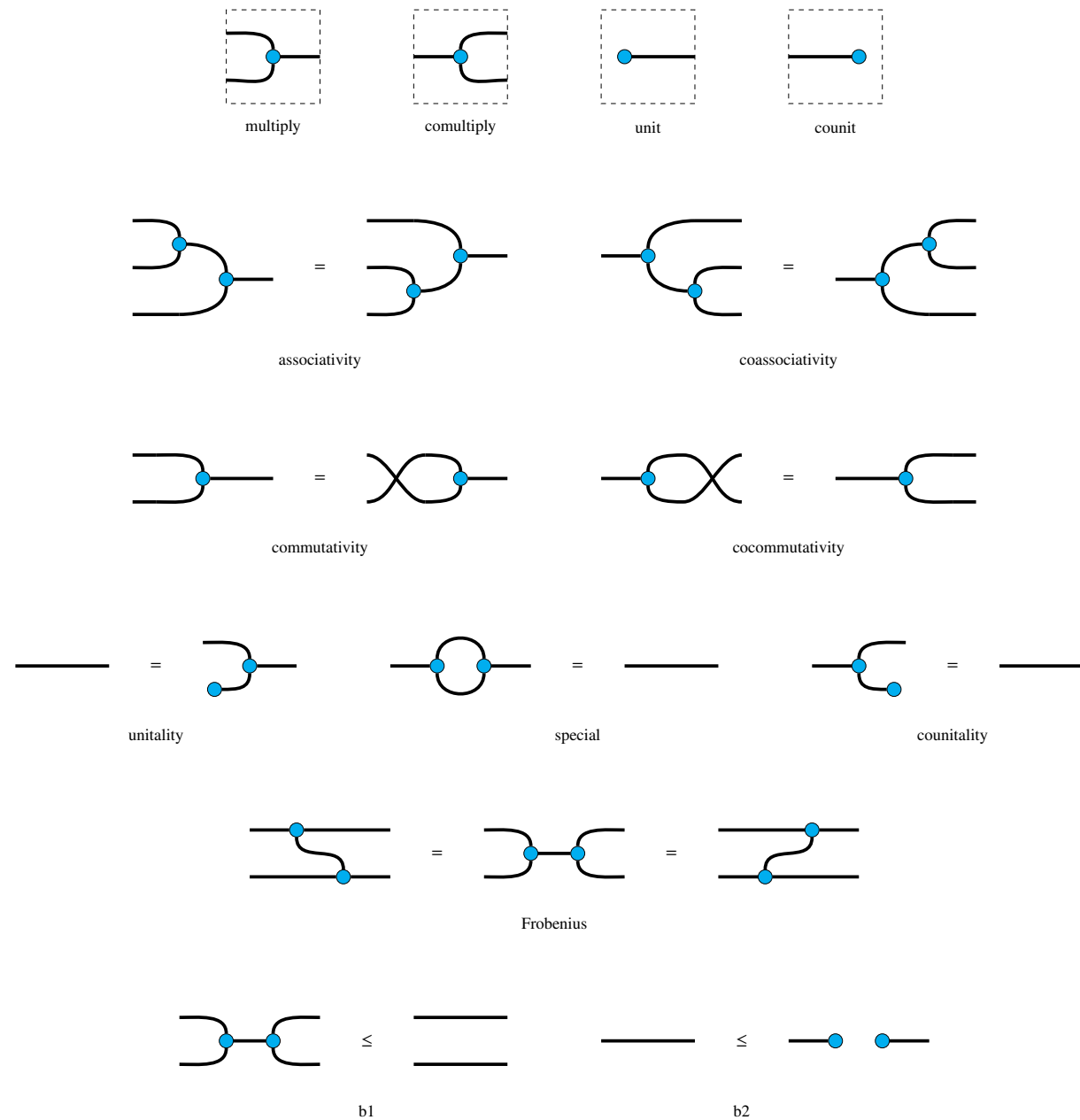
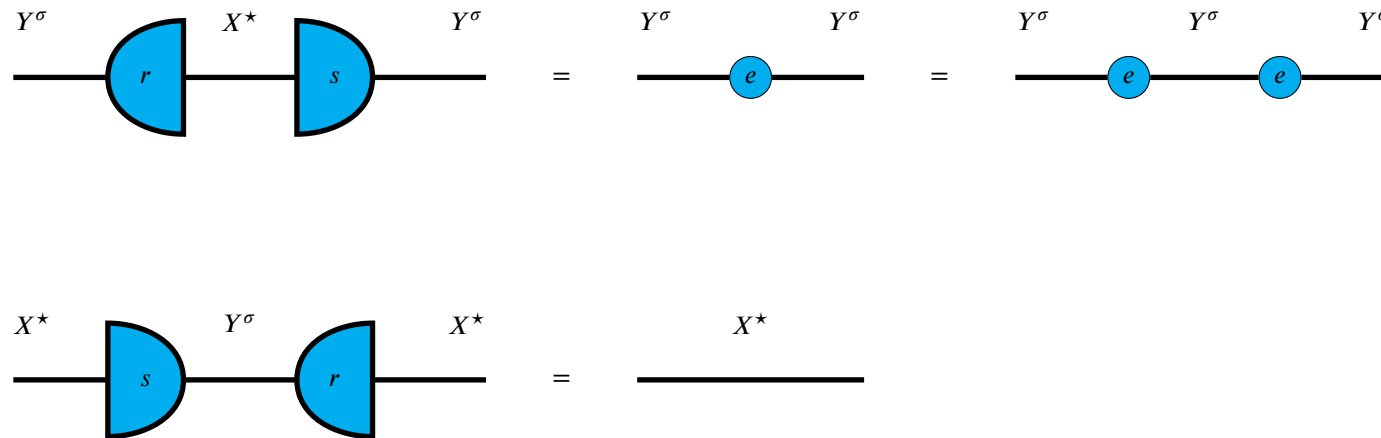


Figure 3.11: The generators (in dashed boxes) and relations that make a spider. When the spider satisfies in addition the three inequalities b1-3, we call it a **relation-spider**.



AN IDEMPOTENT ON Y^σ THAT SPLITS THROUGH A DISCRETE TOPOLOGY X^\star DOES THESE THINGS:

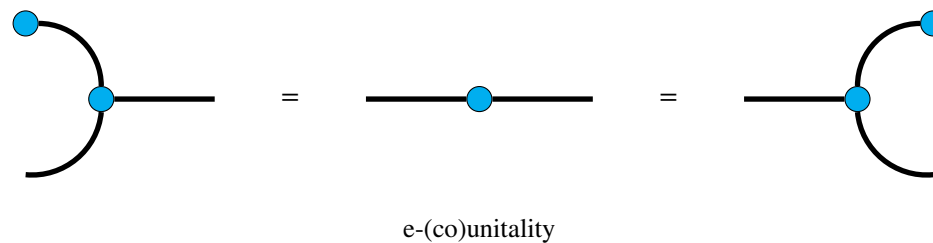
The section: picks a subset of $s(x) \subseteq Y$ for each point $x \in X$. In order to be a split idempotent, $s(x)$ must be distinct for distinct points x .

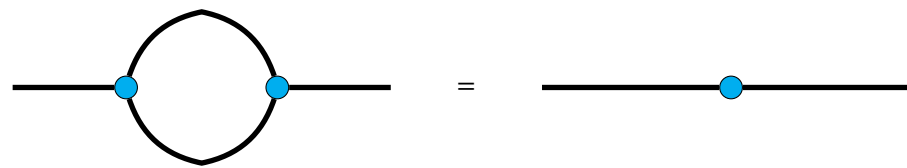
The retract: (reading backwards) picks an open set $r(x) \in \sigma$ for each point $x \in X$. In order to be a split idempotent, $r(x)$ only overlaps $s(x)$ out of all other selected shapes:

$$r(x) \cap s(x') = 1 \iff x = x'$$

The combined effect is that the shapes $s(x)$ become copiable elements of the sticky spider, just as the points are the copiable elements of a regular spider.

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 .



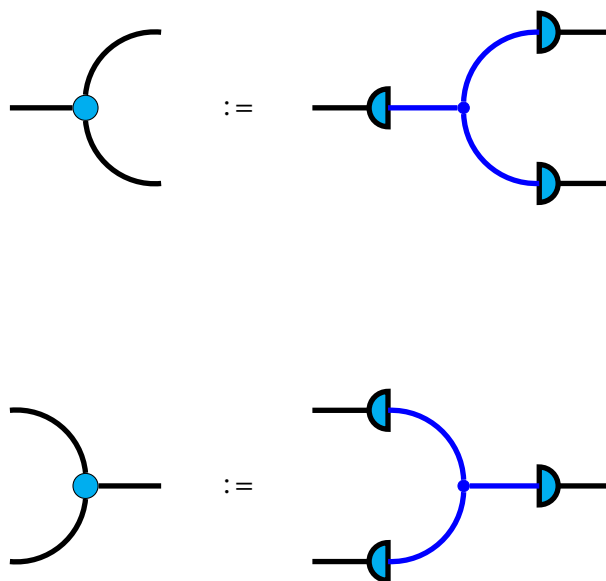


e-special

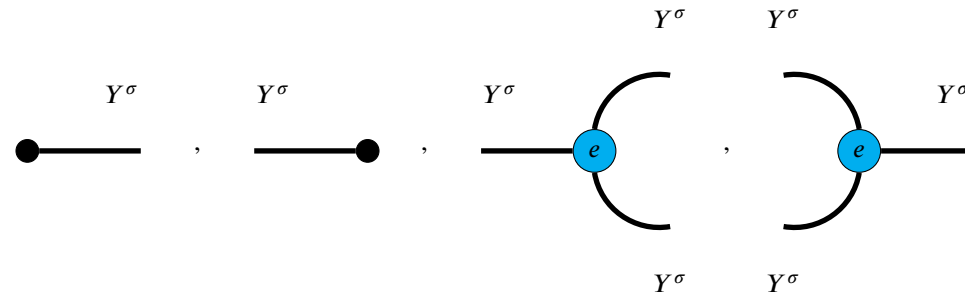
The desired graphical behaviour of a sticky spider is that one can still coalesce all connected spider-bodies together, but the 1-1 spider "sticks around" rather than disappearing as the identity. This is achieved by the following rules that cohere the idempotent e with the (co)unit and (co)multiplications.

placeholder

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

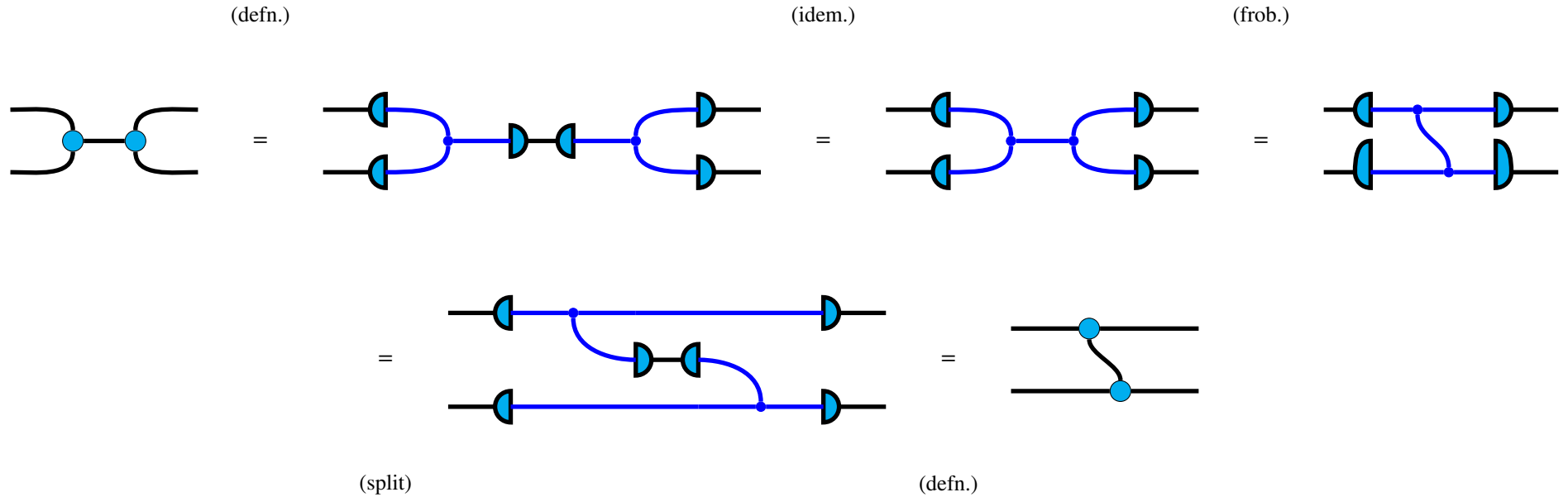


Proposition 3.7.7 (Every idempotent that splits through a discrete topology gives a sticky spider).



is a sticky spider

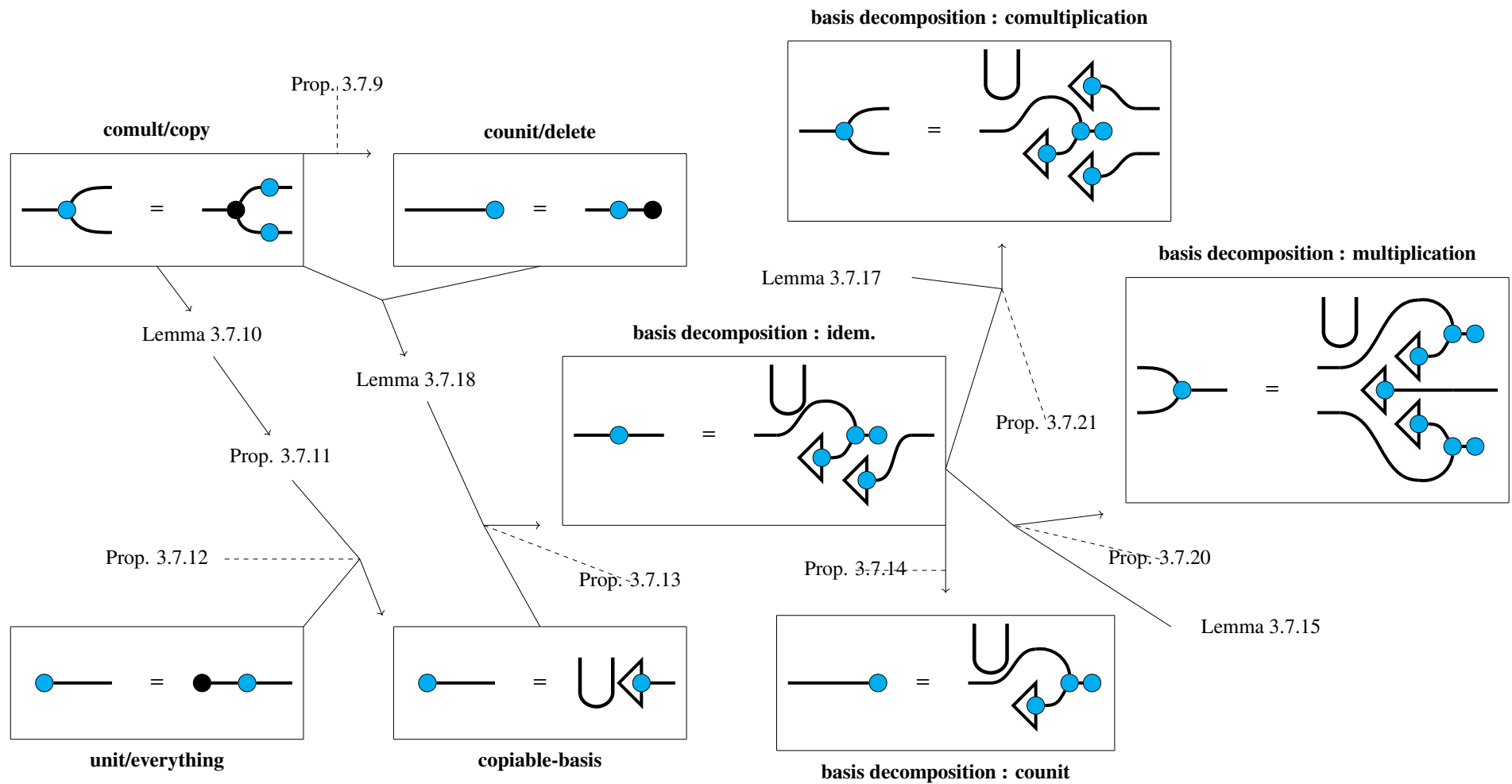
Proof. We can check that our construction satisfies the frobenius rules as follows. We only present one equality; the rest follow the same idea.



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

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

satisfied.



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

$$\begin{array}{c} \text{---} \bullet \text{---} \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array} \Rightarrow \begin{array}{c} \text{---} \bullet \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array}$$

Proof.

$$\begin{array}{c} \text{---} \bullet \text{---} \end{array} \stackrel{\text{(comult/copy)}}{=} \begin{array}{c} \text{---} \bullet \text{---} \end{array} \stackrel{\text{(del)}}{\sqsubseteq} \begin{array}{c} \text{---} \bullet \text{---} \end{array} \\
 \stackrel{\cong}{=} \begin{array}{c} \text{---} \bullet \text{---} \end{array} \stackrel{\text{(e-unit)}}{=} \begin{array}{c} \text{---} \bullet \end{array} \stackrel{\text{(copy-del)}}{\cong} \begin{array}{c} \text{---} \bullet \text{---} \end{array}$$

So:

$$\begin{array}{c} \text{---} \bullet \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array}$$

So:

$$\begin{array}{c} \text{---} \bullet \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array} = \begin{array}{c} \text{---} \bullet \text{---} \end{array}$$

□

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 $e(\{y\}) = \emptyset$ or $e(\{x\}) = e(\{y\})$. Diagrammatically:

$$\begin{array}{|c|} \hline \text{Diagram 1} \in \text{Diagram 2} \Rightarrow \text{Diagram 3} = \text{Diagram 4} \text{ Or } \text{Diagram 5} \\ \hline \end{array}$$

Proof.

Suppose

$$\begin{array}{|c|} \hline \text{Diagram 1} \neq \text{Diagram 2} \\ \hline \end{array}$$

For the claim, we seek:

$$\begin{array}{|c|} \hline \text{Diagram 1} = \text{Diagram 2} \\ \hline \end{array}$$

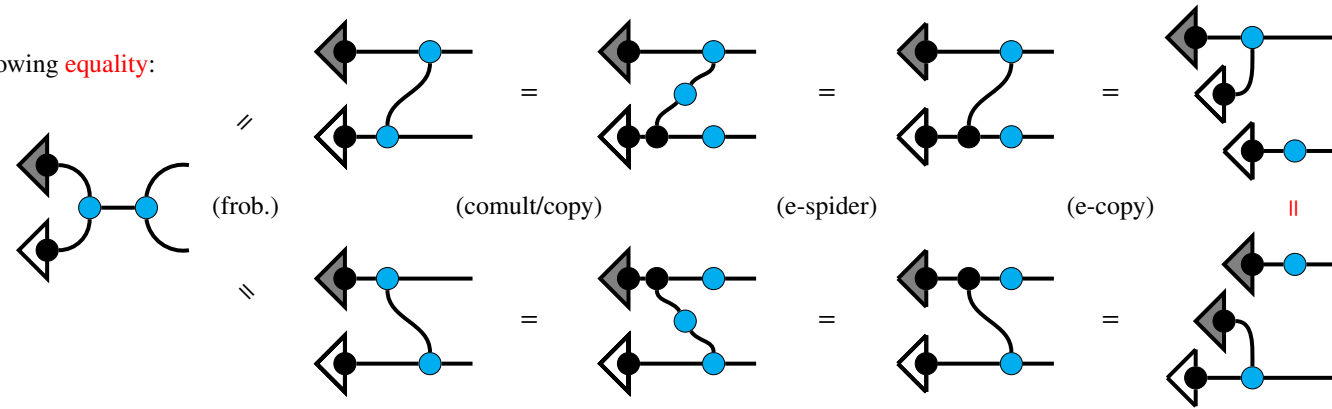
We have the following inclusion:

$$\begin{array}{ccccccc} \text{Diagram 1} & \xrightarrow{\text{(prem.)}} & \text{Diagram 2} & \xrightarrow{\text{(copy)}} & \text{Diagram 3} & \xrightarrow{\text{(comult/copy)}} & \text{Diagram 4} \xrightarrow{\text{(e-special)}} \text{Diagram 5} \\ \text{Diagram 1} & \supseteq & \text{Diagram 2} & = & \text{Diagram 3} & = & \text{Diagram 4} = \text{Diagram 5} \end{array}$$

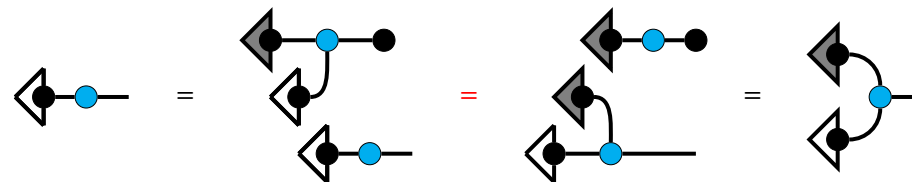
Therefore:

$$\begin{array}{|c|} \hline \text{Diagram 1} = \text{Diagram 2} \neq \text{Diagram 3} \\ \hline \end{array}$$

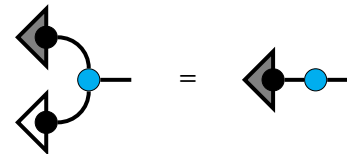
So we have the following **equality**:



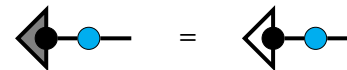
Which implies:



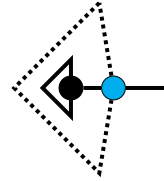
and symmetrically,



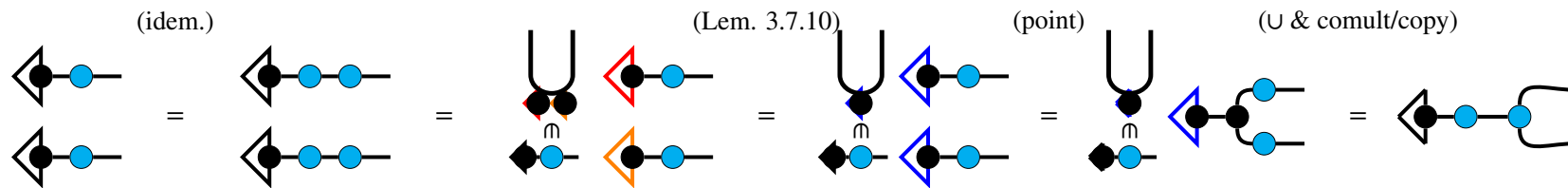
So we have the claim:



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

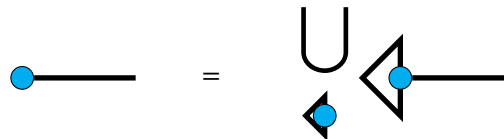


Proof.



□

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.

$$\begin{array}{ccccccc}
 \bigcup_{\forall} \triangleleft \bullet & = & \bigcup_{\forall} \triangleleft \bullet \bullet & \subseteq & \bullet \bullet & = & \bullet \\
 \text{(Lem. 3.7.18)} & & \text{(evr.)} & & \text{(unit/evr.)} & &
 \end{array}$$

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

$$\begin{array}{ccccccc}
 \bullet & = & \bullet \bullet & = & \bigcup_{\forall} \triangleleft \bullet \bullet & = & \bigcup_{?} \triangleleft \bullet \\
 \text{(unit/evr.)} & & & & \text{(Prop. 3.7.11)} & &
 \end{array}$$

So the containment must be an equality.

$$\bullet = \bigcup_{\forall} \triangleleft \bullet$$

□

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

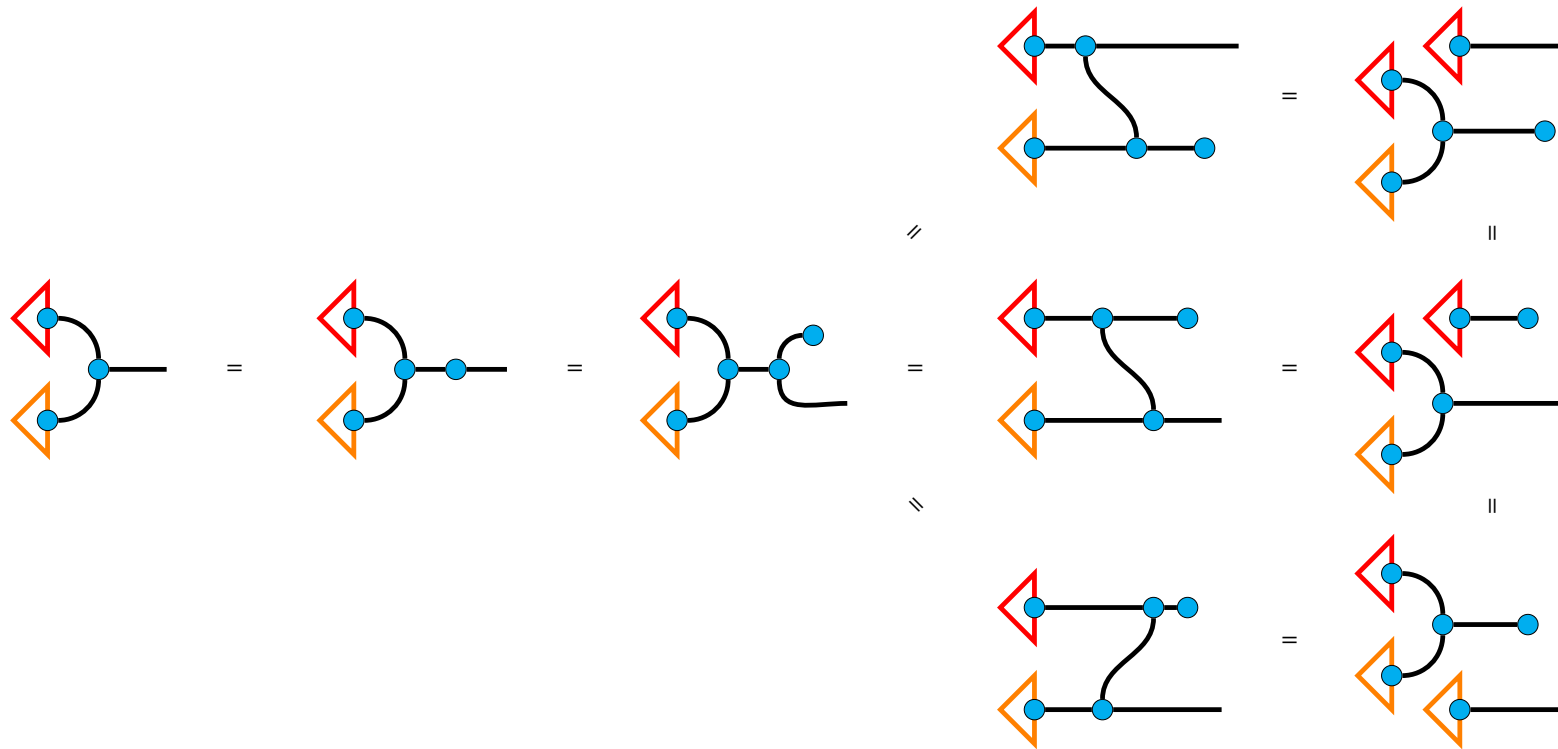
Proof.

(Prop. 3.7.12) (*e*-copiable)

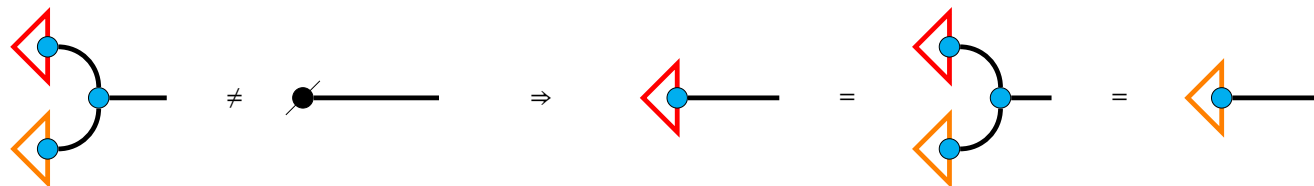
□

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

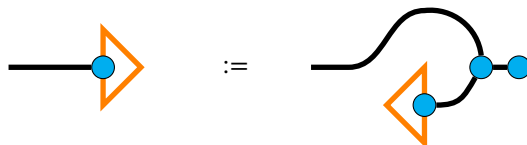
Proof.



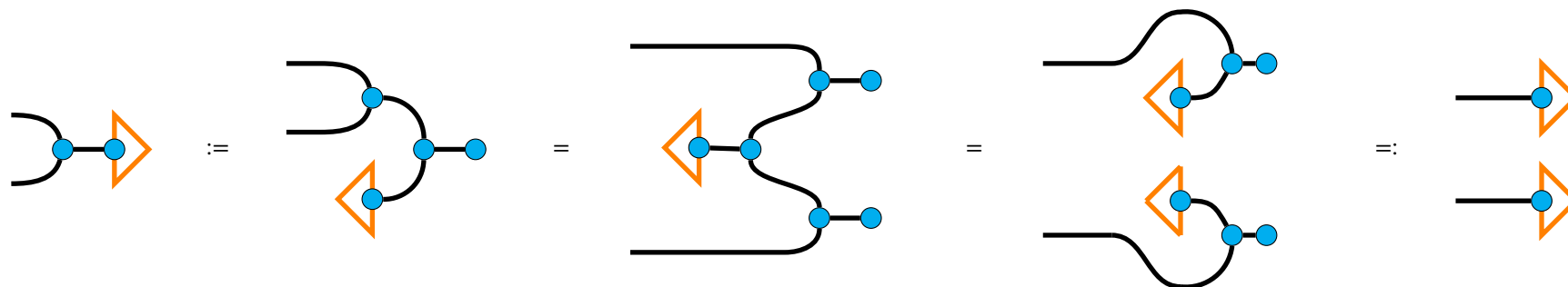
So:



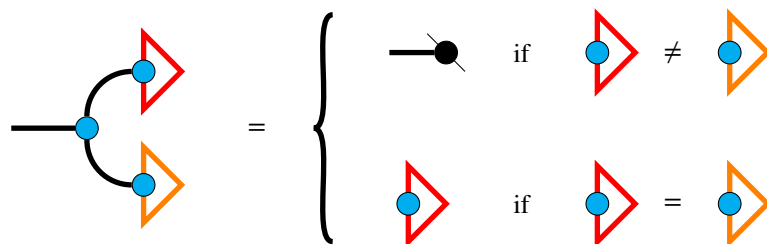
Convention 3.7.16 (Shorthand for the open set associated with an e -copiable). We introduce the following diagrammatic shorthand.



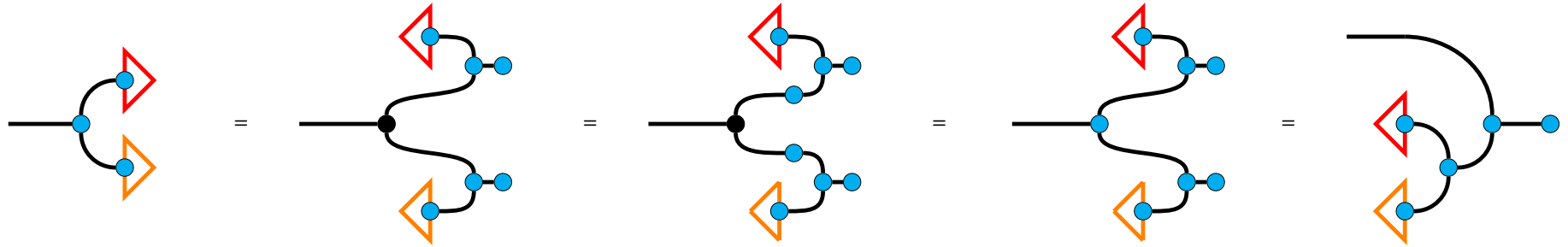
Including the coloured dot is justified, because these open sets are co-copiable with respect to the multiplication of the sticky spider.



Lemma 3.7.17 (Co-match).

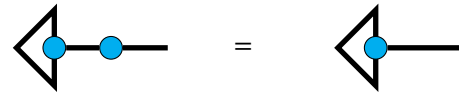


Proof.



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

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

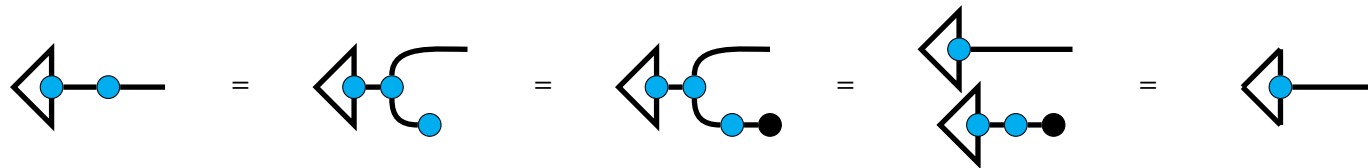


Proof.

(e-counit)

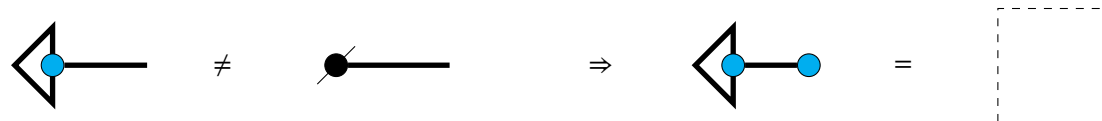
(coun/del)

(e-copy)



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

Lemma 3.7.19 (e-copiables are normal).



Proof.

$$\begin{array}{c}
 \text{(coun/del)} \qquad \qquad \text{(Lem. 3.7.18)} \qquad \qquad \text{(Prem.)} \\
 \triangleleft \bullet \text{---} \bullet = \triangleleft \bullet \text{---} \bullet \text{---} \bullet = \triangleleft \bullet \text{---} \bullet = \square
 \end{array}$$

□

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

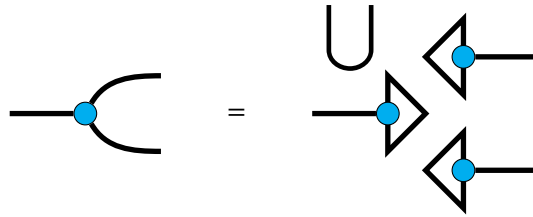
$$\text{---} \cup \text{---} \bullet \text{---} = \text{---} \triangleleft \bullet \text{---} \triangleleft \bullet \text{---} \cup$$

Proof.

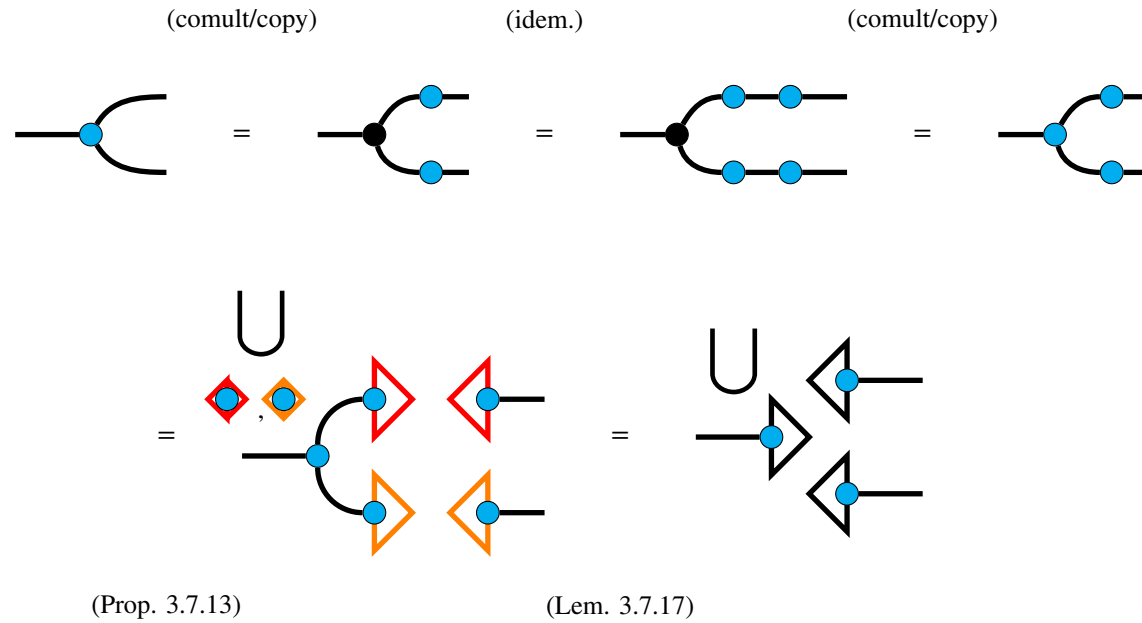
$$\begin{array}{c}
 \text{---} \cup \text{---} \bullet \text{---} = \text{---} \bullet \cup \text{---} \bullet \text{---} = \text{---} \triangleleft \bullet \text{---} \triangleleft \bullet \text{---} \cup \text{---} \triangleleft \bullet \text{---} \triangleleft \bullet \text{---} \cup \\
 \text{(e-spider)} \qquad \qquad \text{(Prop. 3.7.13)} \qquad \qquad \text{(Lem. 3.7.15)}
 \end{array}$$

□

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



Proof.



□

NOW WE CAN PROVE THEOREM 3.7.8.

Proof. The key 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.

placeholder

□

3.8 *Sticky spiders are collections of shapes in space*

...

3.9 *Displacing shapes with symmetries*

Definition 3.9.1 (Topological Group).

Definition 3.9.2 (Sticky spiders equivalent up to symmetric displacement).

3.10 *Moving shapes continuously*

3.11 *Configuration space of a sticky spider*

Definition 3.11.1 (The configuration space of a sticky spider).

Definition 3.11.2 (The "interaction" relation).

Definition 3.11.3 (Movement equivalence classes).

3.12 *Topological models for a selection of concepts*

Construction 3.12.1 (Putting-in and getting-out things from containers).

Construction 3.12.2 (Filling holes, fullness, and fitting).

Construction 3.12.3 (Sending by throwing and conduits).

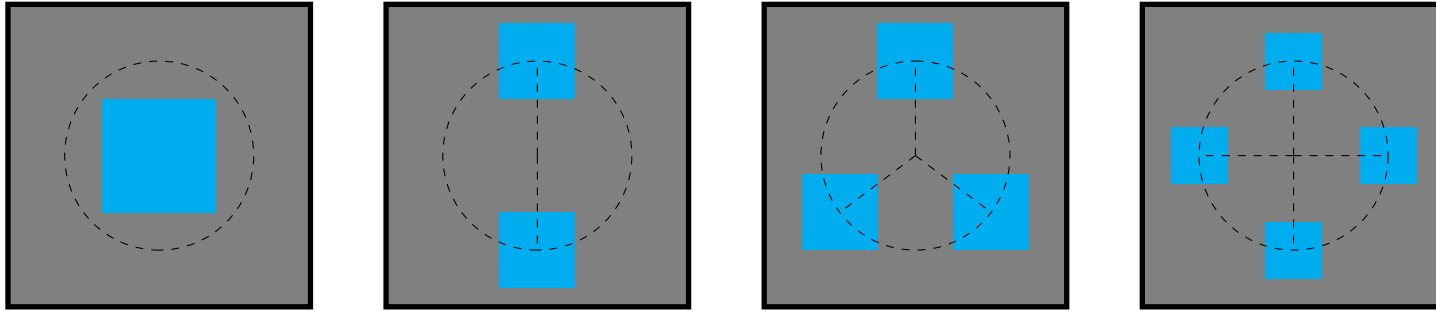
Construction 3.12.4 (Conduit Metaphor).

3.13 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 $\Xi := \blacksquare$ 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; \blacksquare 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 \blacksquare that mimic various compositions of relations.

Proposition 3.13.1 ($((0, 1) \times (0, 1)$ splits through any countable set X). For any countable set X , the open unit square \blacksquare has a sticky spider that splits through X^* .

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.



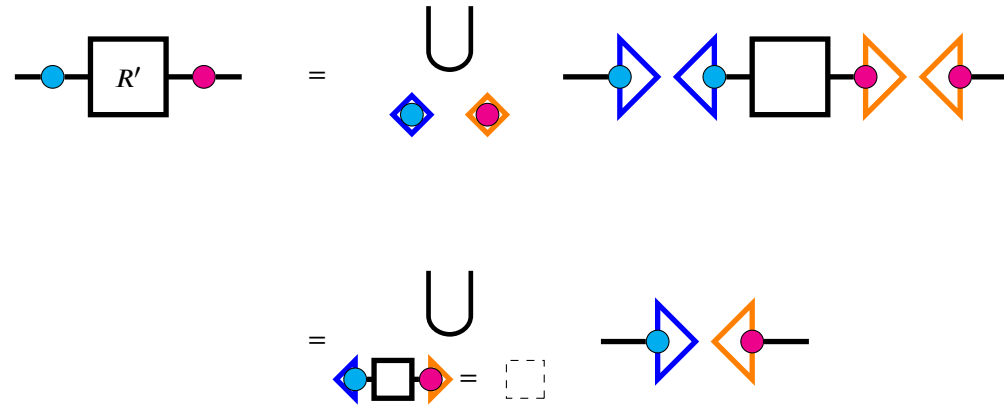
□

Definition 3.13.2 (Morphism of sticky spiders). A morphism between sticky spiders is any morphism that satisfies the following equation.

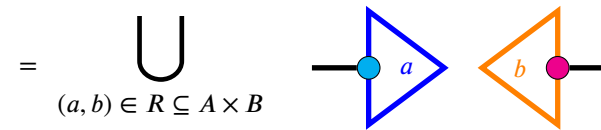
Proposition 3.13.3 (Morphisms of sticky spiders encode relations). For arbitrary split idempotents through A^* and B^* , the morphisms between the two resulting sticky spiders are in bijection with relations $R : A \rightarrow B$.

Proof.

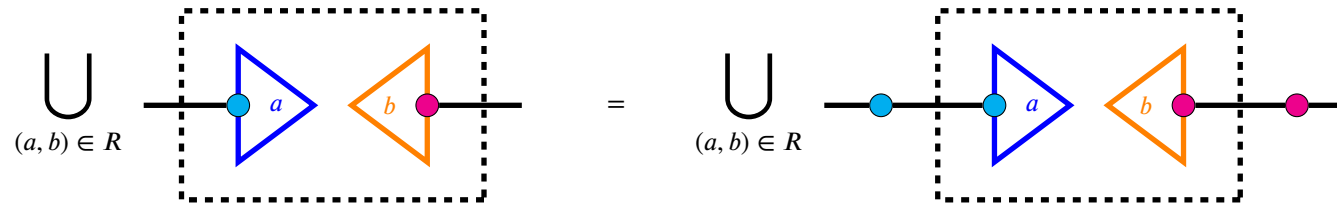
(\Leftarrow) : Every morphism of sticky spiders corresponds to a relation between sets.



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

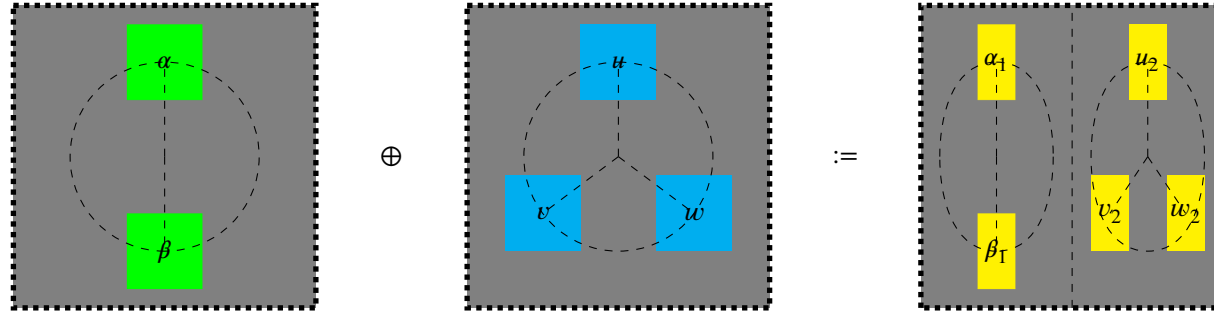


(\Rightarrow) : By idempotence of (co)copiables, every relation $R \subseteq A \times B$ corresponds to a morphism of sticky spiders.



□

Construction 3.13.4 (Representing sets in their various guises within \mathbb{R}). We can represent the direct sum of two \mathbb{R} -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.

$$\begin{array}{cccc}
 \text{---} \text{---} \text{---} & \text{---} \text{---} \text{---} & \text{---} \text{---} \text{---} & \text{---} \text{---} \text{---} \\
 (x, y) \mapsto (\frac{x}{2}, y) & (x, y) \mapsto (\frac{x+1}{2}, y) & (x, y)|_{x < \frac{1}{2}} \mapsto (2x, y) & (x, y)|_{x > \frac{1}{2}} \mapsto (2x - 1, y)
 \end{array}$$

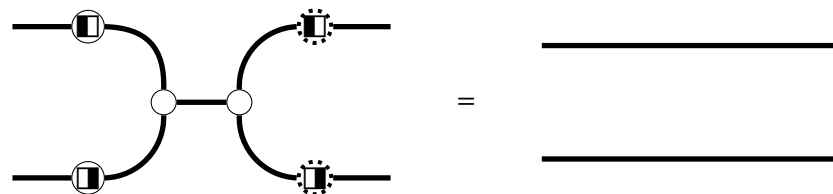
We express the ability of these relations to encode and decode the unit square in just either half by the following graphical equations.

$$\text{---} \text{---} \text{---} = \text{---} = \text{---} \text{---} \text{---}$$

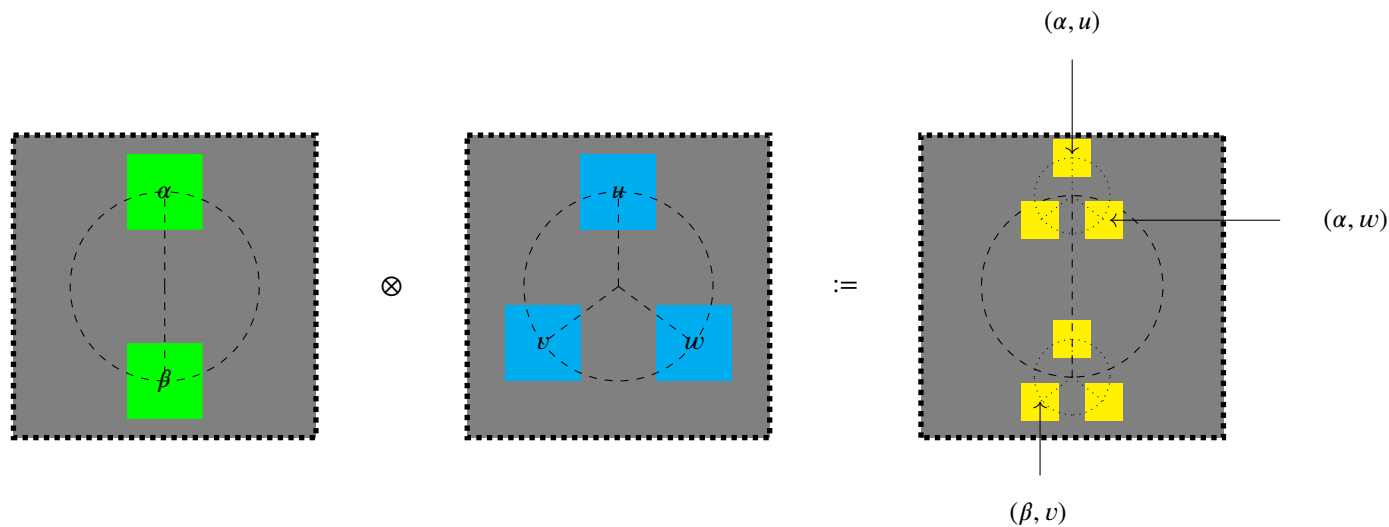
Now, to put the two halves together and to take them apart, we introduce the following two relations. In tandem with the squishing and stretching we have defined, these will behave just as the projections and injections for the direct-sum biproduct in **Rel**.

$$\begin{array}{ccc}
 \text{---} \text{---} \text{---} & := & \text{---} \text{---} \text{---} \cup \text{---} \text{---} \text{---} \\
 \text{---} \text{---} \text{---} & := & \text{---} \text{---} \text{---} \cup \text{---} \text{---} \text{---}
 \end{array}$$

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

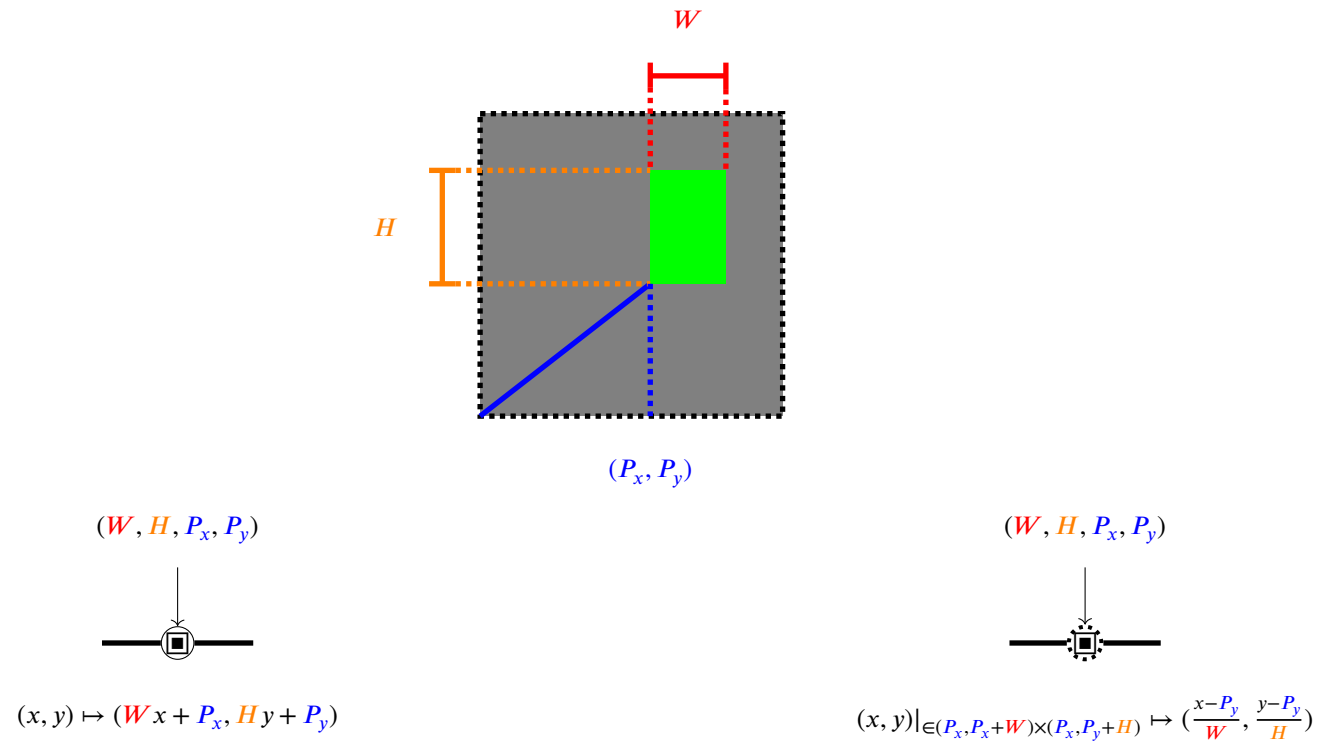


We encode the tensor product $A \otimes B$ of representations by placing copies of B in each of the open boxes of A .



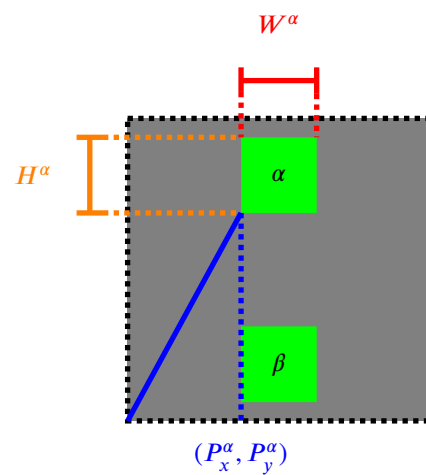
The important bit of technology here is a family of homeomorphisms of \mathbb{B} parameterised by axis-aligned open boxes. We depict the parameters outside the body of the homeomor-

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



Now, for every representation of a set in \mathbb{R}^2 by a sticky-spider, where each element corresponds to an axis-aligned open box, we can associate each element with a squish-stretch

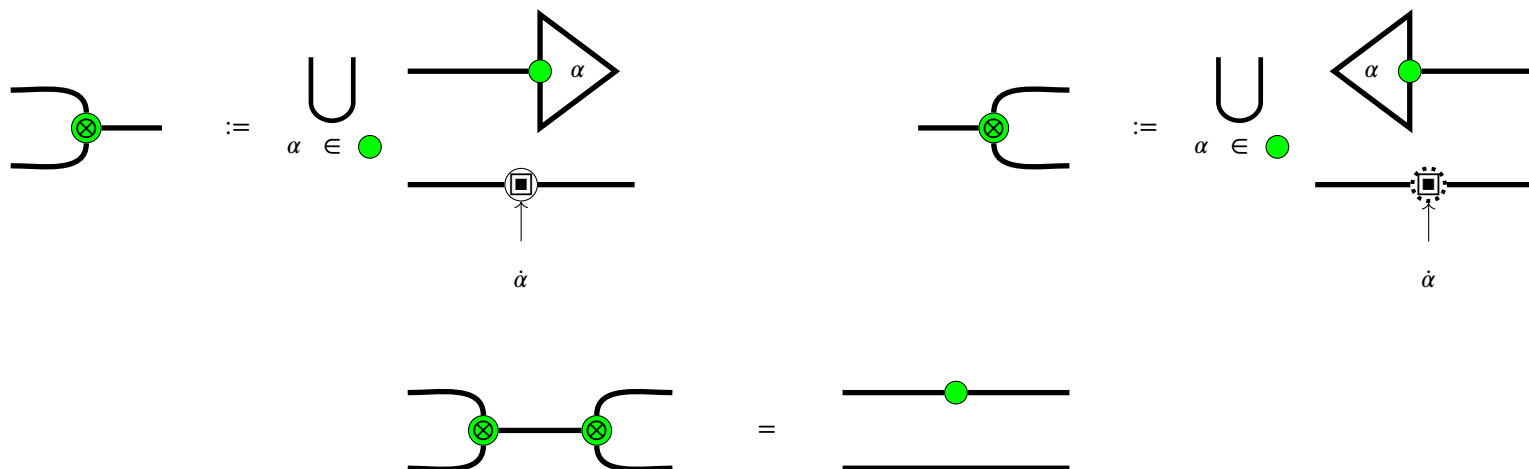
homeomorphism via the parameters of the open box, which we notate with a dot above the name of the element.



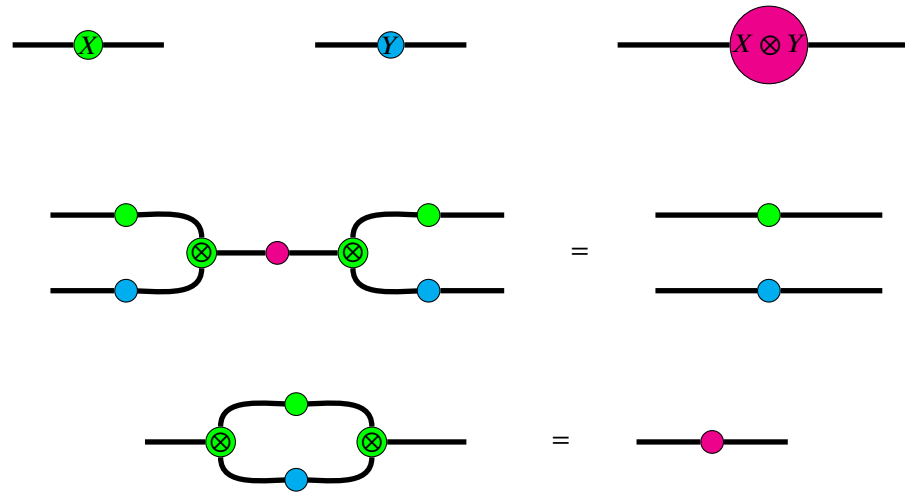
$$\dot{\alpha} = (W^\alpha, H^\alpha, P_x^\alpha, P_y^\alpha)$$

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

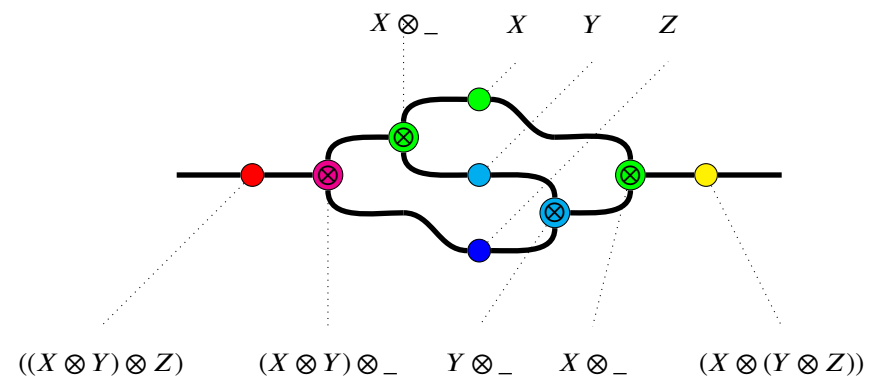
Now we can define the "tensor X on the left" relation $_ \rightarrow X \otimes _$ and its corresponding cotensor.



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

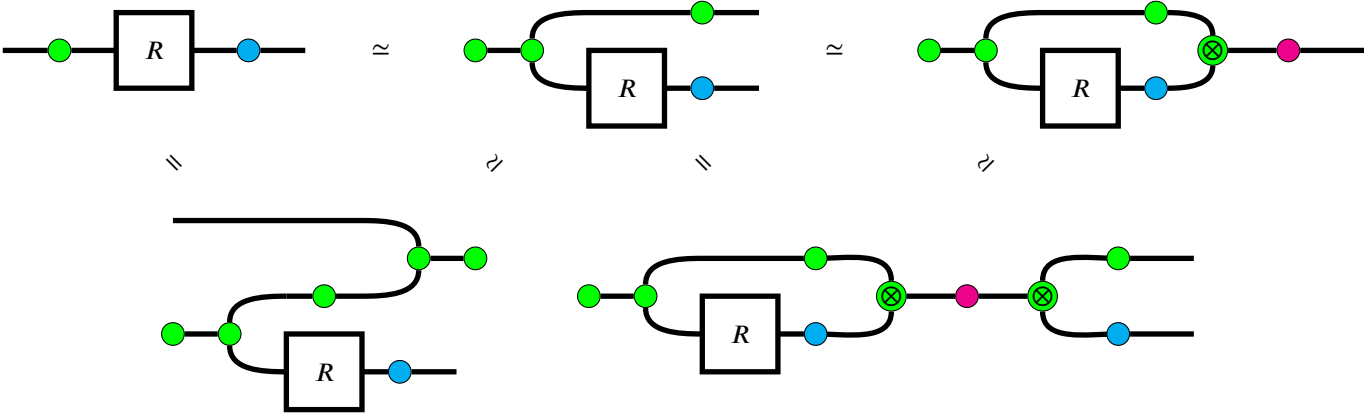


And by construction, the (co)tensors and (co)pluses interact as we expect, and they come with all the natural isomorphisms between representations we expect. For example, below we exhibit an explicit associator natural isomorphism between representations.

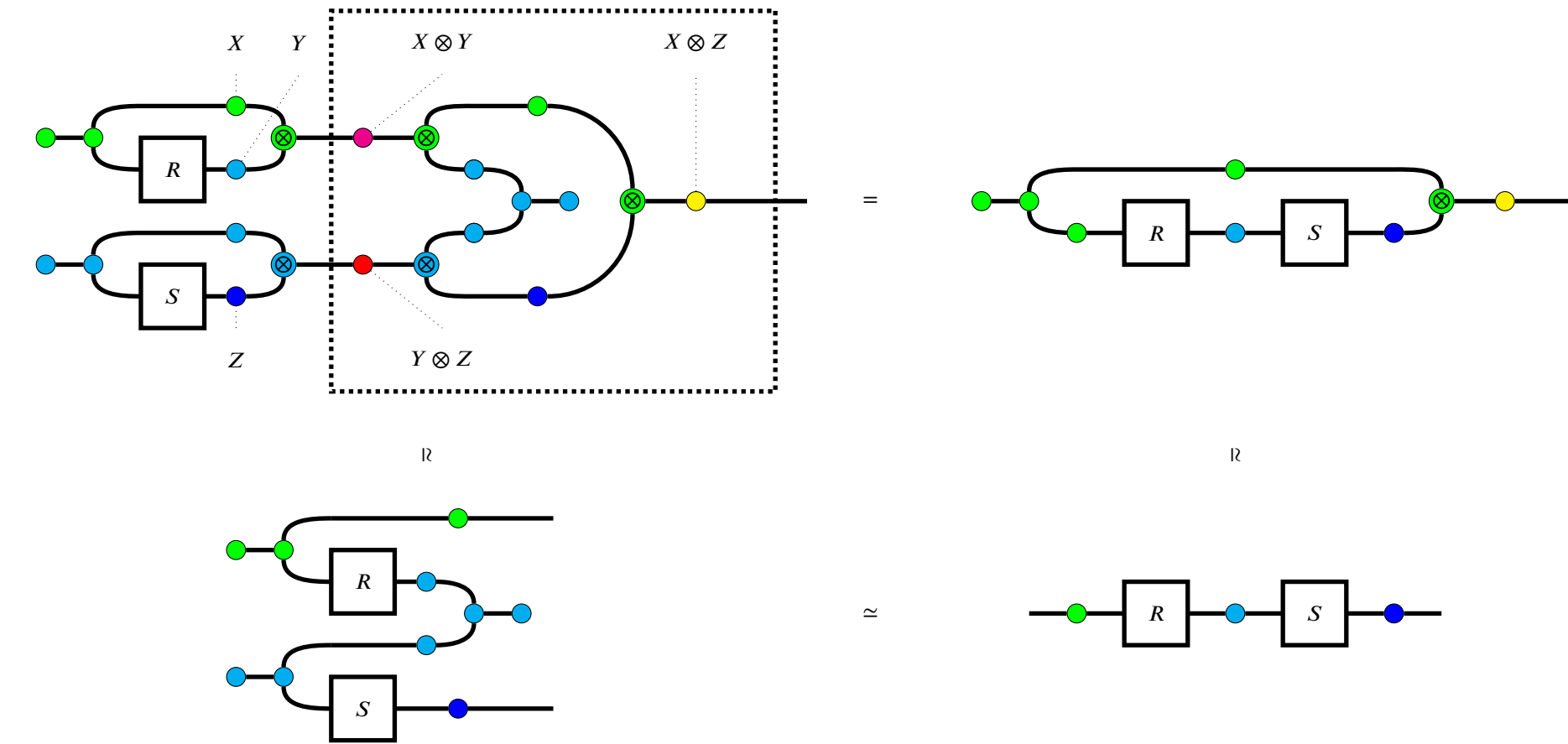


Construction 3.13.5 (Representing relations between sets and their composition within \mathbb{M}). With all the above, we can establish a special kind of process-state duality; relations as

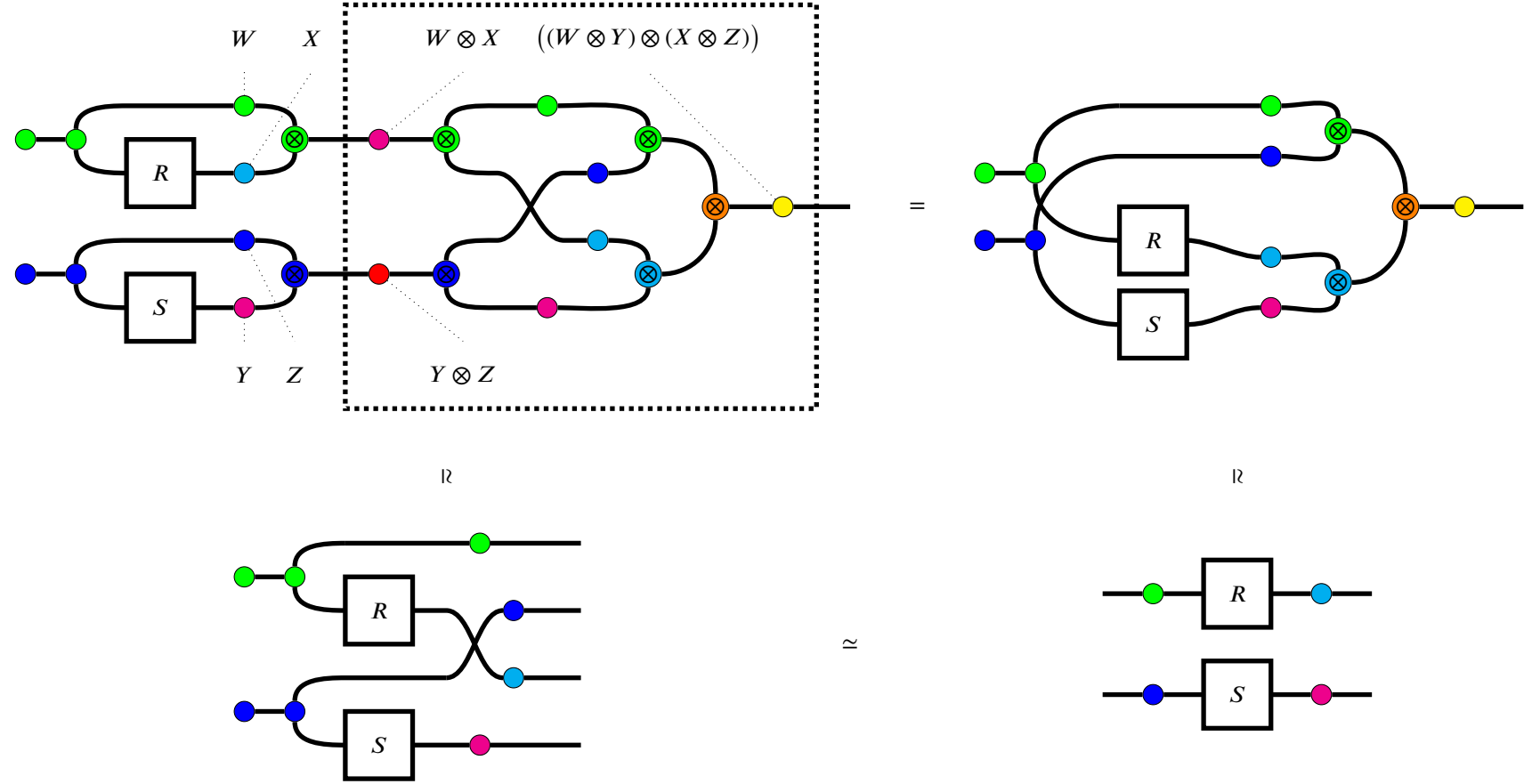
processes are isomorphic to states of \mathbb{M} , up to the representation scheme we have chosen.



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



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



4

Metaphor, narrative, other pictures

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

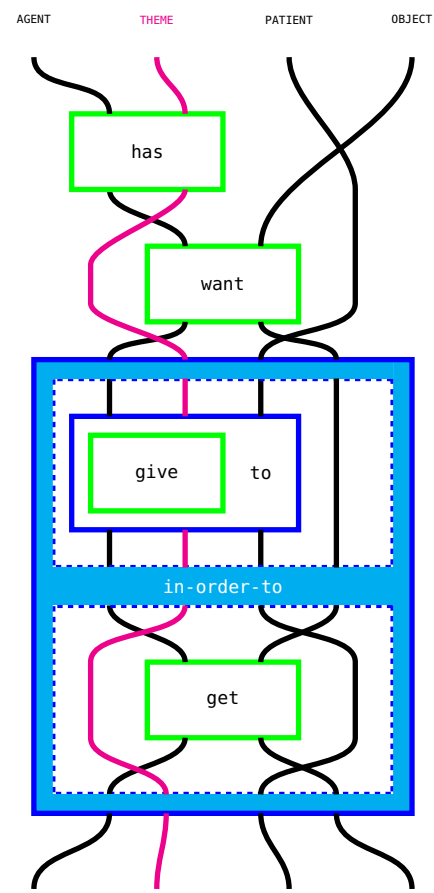
"Do you have time to look at this?"

"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