Graph Rewrite Systems for Classical Structures in †-Symmetric Monoidal Categories

Aleks Kissinger

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Master of Science

September 2008

Abstract

This paper introduces several related graph rewrite systems derived from known identities on classical structures within a †-symmetric monoidal category. First, we develop a rewrite system based on a single classical structure, and use it to develop a proof of the so-called “spider-theorem,” where a connected graph containing a single classical structure can be uniquely determined by the number of inputs and outputs (i.e. it can be rewritten as a graph resembling an n-legged spider). These spiders are shown to be the normal forms of graphs containing a single classical structure. Next, complementary classical structures are introduced, as well as a new rewrite system on graphs of red and green spiders. A proof of convergence is given for a limited two-colour rewrite system, as well as insights into ways to approach normalisation in a more powerful rewrite system.

Acknowledgements

I would first like to thank my supervisor Ross Duncan. It has been a privilege working with him for the past six months, as his knowledge and energy have been inspirational to me and pivotal in the creation of this work. To him I give my deepest thanks for the amount of time he has devoted to helping me, especially given the timing of this project.

For their help and ideas, I would like to thank Lucas Dixon at the University of Edinburgh and Simon Perdrix, as well as all of the other members of the quantum computing research group at Oxford. Among these people, I would especially like to thank Bob Coecke and Samson Abramsky. Their lectures and kind guidance led me into this program and are largely to credit for my excitement about the subjects represented in this work.

My thanks also go to Michael Collins, Luke Ong, Margaret Sloper, Wendy Adams, Sandy Patel, and everyone else who has worked to make the MFoCS program at Oxford so extraordinary.

My most heartfelt thanks goes to John Hale at the University of Tulsa. He has been a friend, mentor, and inspiration to me. Without him, I most surely would not be where I am today. Lastly, I would like to thank my parents, for more than I could possibly list here.

Contents

Chapter 1

Introduction

When reasoning about the behaviour of a quantum system, it is important to adopt mechanisms that cope with the inherently incomplete knowledge one has of the system. More specifically, reasoning about quantum systems requires one to distinguish information that is "classical" in nature from information that is somehow "quantum." Classical information is perfectly knowable. One could write it down, or erase it, or make many copies of it. In quantum terms, this information corresponds to the possible outcomes of a measurement. The other kind of information is only imperfectly knowable. This is the information associated with a quantum state that is in a superposition of more than one classical state. This interpretation yields the natural representation of a quantum state as a Hilbert space, where the classical states are a basis, measurements are projections and state evolutions are unitaries. Though such an interpretation is quite natural, it soon becomes cumbersome [7] when one seeks to grasp the essence of the interplay between quantum and classical information. Following this observation, much work in recent years has gone into seeking suitable abstractions of the Hilbert space interpretation of quantum information [1, 6]. Such abstractions must be both simple enough to be practical for large systems and powerful enough to recover many of the uniquely "quantum" qualities of the computation.

To this end, †-symmetric monoidal categories (†-SMCs) provide a clean and powerful means of reasoning about quantum computation [1]. Since morphisms in symmetric monoidal categories are inherently two-dimensional (tensor $\times$ composition), graphs often provide an elegant means of expressing and manipulating them (see [1, 9, 19, 14, 13]). This project is concerned with †-SMC's generated from a single object and one or more **classical structures** [9]. A classical structure is a pair of morphisms that "copy" and "delete" a quantum observable. If two quantum observables are mutually unbiased (i.e. the first observable is an equal superposition of the second), we call their associated classical structures complementary. Complementary classical structures provide a powerful computational primitive [8]. We explore the decidability of equations resulting from the axioms of classical structures by introducing a series of graph rewrite systems and showing that they are well-behaved.

Classical structures are highly regular when they are composed with themselves, to the extent that any such configuration is uniquely determined by the number of inputs and outputs. Expressed graphically, any configuration of the morphisms from a single classical structure can be rewritten as a graph resembling an n-legged spider. This result is stated without proof in [8] and others, so the first aim of this work is to provide context and a simple proof of this "spider theorem" using a graph rewrite system on a single classical structure. After the spider theorem is proven, we go on to study graphs of complementary classical structures. In the next rewrite system, we introduce a second classical structure and add rules representing the spider theorem and the interaction between the classical structures. We show convergence of a subset of this rewrite system by critical pair analysis. We conclude by showing the non-termination of the full system and offering possible approaches to a proof of confluence.

At the time of this writing, we are undergoing work on an automated tool for rewriting quantum graphs. This paper aims to show the correctness and feasibility of such a tool.

1.1 Related Work

Monoidal categories are inherently two-dimensional. In addition to composing maps in the normal (vertical) sense, one can also lay them "side-by-side" via the tensor product. This spatial interpretation of morphisms yields very natural graphical representations. In 1980, Kelly and Laplaza showed a strong correlation between the axioms of a type of monoidal category called a **compact closed category** and the concept of graph isomorphism [18]. Abramsky and Coecke showed in 2004 that strongly compact closed categories, as well as their graphical representations, provide a clean semantic structure for describing and manipulating quantum protocols [1]. In 2005, Selinger formalised this correlation and showed proofs for the validity of graphical representations of several related kinds of symmetric monoidal categories, including †-SMC's, which provide the setting for this work. Still using a graphical language, Coecke and Duncan showed that a particular class of related morphisms, called complementary classical structures, act as an ideal computational primitive for many quantum protocols. In doing so, they illustrated all of the identities upon which the rewrite systems in this paper are based.

Term rewrite systems have a rich history, primarily in studying computation. For an arbitrary term rewrite system, deciding confluence and termination was shown to be equivalent to the word problem. Because of this, much work in the past fifty years has gone into identifying sufficient properties for confluence and termination. Many influential results came from Church and Rosser's study of λ-calculus in the first part of the 20th century [5]. When considered as a term rewrite system, the untyped λ-calculus is was shown to be confluent but not terminating. In 1942, Maxwell Newman found that locally confluent, terminating rewrite systems are confluent [12]. Huet shows in [17] that a certain kind of local confluence, called strong local confluence implies strong confluence, which as the name suggests, is a stronger property than confluence. Since then Bezem, von Oostrom, Gramlich and others sought weaker sufficient conditions for confluence [4, 15], particularly in systems where rewrite rules have minimal overlap.

Unlike term rewrite systems, there is no single canonical formalism for graph rewrite systems. There are categorical approaches, which treat rewrites as pullbacks in a category with graphs as objects and graph homomorphisms as arrows [10]. These maintain context using a gluing graph, or a sub-graph shared by the left and right hand side of a rewrite rule that maintains connections after the rewrite is performed. There are also approaches that treat graphs as adjacency matrices, and rewrites as matrix transformations [20]. Another technique for graph rewriting (as seen in [11]), is to deal with hypergraphs, where the context of a rewrite is stored as a single hyperedge, connecting and ordering source vertices of a rewrite rule. The technique in this paper is a combination of de Micheaux's and that in [13], in which substitutions are built on graph homomorphisms with several consistency properties (called **matchings** in [13] and **occurrences** in [11]) though we remove the complexity of ordering boundary vertices by insisting beforehand that graph components be commutative.

1.2 Symmetric Monoidal Categories

A **monoidal category** is a category equipped with a bifunctor $-\otimes-$ called the tensor product, a unit object I, and the following natural isomorphisms: $\lambda_A : A \to I \otimes A$, $\rho_A : A \to A \otimes I$, and $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$. These correspond respectively to left unit, right unit, and associativity, and they obey certain natural coherence conditions. A **symmetric monoidal category** contains an additional natural isomorphism $\sigma_{A,B} : A \otimes B \to B \otimes A$ that is coherent with the monoidal structure. For more detail and a precise definition of "coherence," see e.g. [2].

A **compact closed category** is a symmetric monoidal category where each object A has a dual object A^* and a pair of mappings $d_A : I \to A^* \otimes A$ and $e_A : A \otimes A^* \to I$, such that the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "\rho_A"] \arrow[d, "id_A"] & A \otimes I \arrow[r, "id_A \otimes d_A"] & A \otimes (A^* \otimes A) \arrow[d, "\alpha_{A,A*,A}"] \\
A & I \otimes A \arrow[l, "\lambda_A^{-1}"] & (A \otimes A^*) \otimes A \arrow[l, "e_A \otimes id_A"]
\end{tikzcd} \tag{1.1}$$

$(-)^*$ extends to a contravariant functor by letting $f^* : B^* \to A^*$ be defined as:

$$f^* = \rho_{A^*}^{-1} \circ (id_{A^*} \otimes e_A) \circ (id_{A^*} \otimes f \otimes id_{B^*}) \circ (d_A \otimes id_{B^*}) \circ \lambda_{B^*}$$

$(-)^*$ preserves the monoidal structure and has a natural isomorphism $A \cong A^{**}$ [18, 19].

A **†-symmetric monoidal category** (or †-SMC) is an SMC with an additional identity-on-objects contravariant endofunctor $(-)^\dagger : \mathcal{C} \to \mathcal{C}^{op}$. It also obeys coherence conditions, such as $(f \otimes g)^\dagger = f^\dagger \otimes g^\dagger$. For details, see [19], [8], or [1].

A **†-compact closed category**, as the name suggests, is a †-SMC that is compact closed. We also require that $\sigma_{A,A^*} \circ e_A^\dagger = d_A$.

Definition 1.2.1. A **classical structure** is a cocommutative comonoid $(A, \delta_Z : A \to A \otimes A, \epsilon_Z : A \to I)$ where δ is isometric and frobenius, meaning:

$$\delta_Z^\dagger \circ \delta_Z = id_A \quad \text{and} \quad \delta_Z \circ \delta_Z^\dagger = (\delta_Z^\dagger \otimes id_A) \circ (id_A \otimes \delta_Z)$$

Remark 1.2.2. Since $(-)^\dagger$ is a contravariant functor, $(A, \delta_Z^\dagger, \epsilon_Z^\dagger)$ forms a commutative monoid.

A pair of classical structures $(A, \delta_Z, \epsilon_Z)$ and $(A, \delta_X, \epsilon_X)$ are known as **complementary** if they satisfy a number of properties described in detail in [8]. There are three properties we are interested in here. The first is $\epsilon_Z^\dagger \circ \epsilon_X = \epsilon_X^\dagger \circ \epsilon_Z = k$, where $k : I \to I$ is a special arrow called a scalar, by analogue to Hilbert spaces. The second property is:[1]

$$k \otimes (\delta_Z \circ \epsilon_X^\dagger) = id_I \otimes (\epsilon_X^\dagger \otimes \epsilon_X^\dagger) \quad \text{and} \quad k \otimes (\delta_X \circ \epsilon_Z^\dagger) = id_I \otimes (\epsilon_Z^\dagger \otimes \epsilon_Z^\dagger)$$

In other words, the comultiplication from one classical structure copies the unit of the other.

[1]This property is usually given differently, employing an abstract analogue to scalar multiplication: $k \bullet \phi$. This construction is not necessary here, so we refer the interested reader to [1].

The third property is:

$$k \otimes \Big((\delta_X^\dagger \otimes \delta_X^\dagger) \circ (id_A \otimes \sigma_{A,A} \otimes id_A) \circ (\delta_Z \otimes \delta_Z) \Big) = id_I \otimes (\delta_Z \circ \delta_X^\dagger)$$

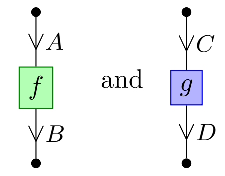
This is called the bi-algebra property, because corresponds to the coherence of multiplication with comultiplication in a bialgebra. [8] shows that complementary classical structures do indeed form a scaled bi-algebra.

Theorem 1.2.3. *The †-SMC generated by one or more classical structures on a single, self dual object $A = A^*$ is †-compact closed.*

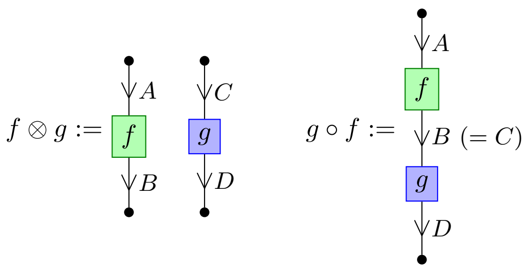
Proof. Let $d_A = \epsilon_Z^\dagger \circ \delta_Z$ and let $e_A = d_A^\dagger$. Since δ_Z is cocommutative, $d_A = e_A^\dagger = \sigma_{A,A} \circ e_A^\dagger$. The commutation of diagram (1.1) follows from the frobenius and comonoidal properties of the classical structure. The derivation is omitted here, but we see it graphically in the proof of theorem 2.2.1. □

1.2.1 SMC Morphisms as Graphs

Consider two arrows $f : A \to B$ and $g : C \to D$ in a †-compact closed category. We can represent these graphically as boxes with labelled edges:



We can express the tensor as juxtaposition, and arrow composition as graph composition (assuming $B = C$):

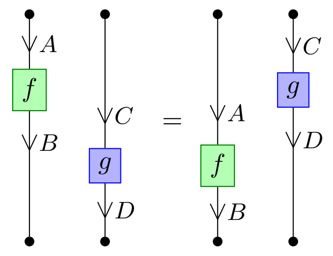


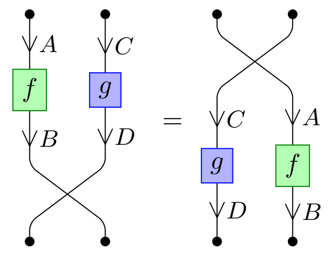
Remark 1.2.4. All graphs in this paper should be read top-to-bottom, as in the previous examples.

Since $- \otimes -$ is a bifunctor and σ is a natural transformation:

$$(id_B \otimes g) \circ (f \otimes id_C) = (f \otimes id_D) \circ (id_A \otimes g) \quad \text{and} \quad \sigma_{A,C} \circ (f \otimes g) = (g \otimes f) \circ \sigma_{B,D}$$

These have graphical representations

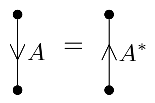




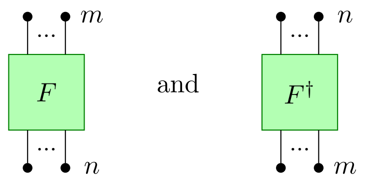
These observations suggest that there is a relationship between the axioms of a †-compact closed category and graphs. This is formalised in the following theorem by Selinger [19].

Theorem 1.2.5. *(Graphical language of dagger compact closed categories) A well-typed equation between morphisms in the language of †-compact closed categories follows from the axioms of †-compact closed categories if and only if it holds, up to graph isomorphism, in the graphical language.*

Dual objects in a compact closed category provide a means of flipping the arrow direction. We define:



If the category is generated by a single, self-dual object, we often omit edge labels and directions. We represent tensor types as multiple inputs and outputs. For $F : A^{\otimes m} \to A^{\otimes n}$, we have:



If all such boxes represent morphisms that are commutative and cocommutative, then we need not distinguish incident edges (though we must still distinguish incoming and outgoing edges). Therefore, it suffices to represent a morphism composed of such boxes and natural isomorphisms from the †-compact closed structure as a general undirected graph. By theorem 1.2.3, we can use an undirected graph to represent any morphism in the †-SMC generated by one or more classical structures on a single self-dual object.

1.3 Abstract Reduction Systems

It is often useful to study properties of reduction systems without reference to the type of objects being reduced. For that purpose we introduce abstract reduction systems [17].

Definition 1.3.1. An **abstract reduction system** (ARS) $\mathcal{A}$ is a pair $(A, \to)$. Where A is a set and $\to$ is a binary relation, called the "reduction relation."

For a relation $\rightarrow$, we can define $\leftarrow$ as the inverse, $\leftrightarrow$ as the symmetric closure, and $\xrightarrow{*}$ transitive, reflexive closure.

1.3.1 Termination

Definition 1.3.2. For an ARS $\mathcal{A}$, any $a \in A$ is said to be a normal form if there exists no a' such that $a \rightarrow a'$.

Definition 1.3.3. A reduction relation $\rightarrow$ is **terminating** if $\leftarrow$ is well-founded. I.e. there exists no infinite chain $a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots$.

Some literature (e.g. [17]) refers to terminating relations as "Noetherian relations," after the mathematician Emmy Noether. Termination guarantees that any reduction sequence will terminate in a (not necessarily unique) normal form.

One often proves termination by identifying a reduction order.

Definition 1.3.4. A **reduction order** $\prec$ on an ARS $\mathcal{A}$ is a pre-order that is strict, well-founded, and compatible with the rewrite system. That is to say, for every $a, b \in A$, $a \rightarrow b \Rightarrow a \succ b$.

Since $\rightarrow \subseteq \succ$, we have $\leftarrow \subseteq \prec$. Since $\prec$ is well founded, $\leftarrow$ is well-founded. So $\rightarrow$ is terminating.

1.3.2 Confluence

One of the most important properties of well-behaved reduction relations is confluence.

Definition 1.3.5. A reduction $\rightarrow$ is **confluent** iff

$$\forall a, b, c \in A.(b \xleftarrow{*} a \xrightarrow{*} c \Rightarrow \exists d \in A.(b \xrightarrow{*} d \xleftarrow{*} c))$$

Some literature instead uses the (equivalent) notion of being Church-Rosser:

Definition 1.3.6. A reduction $\rightarrow$ is **Church-Rosser** (C-R) iff

$$\forall a, b \in A.(a \overset{*}{\leftrightarrow} b \Rightarrow \exists c \in A.(a \xrightarrow{*} c \xleftarrow{*} b))$$

Definition 1.3.7. An ARS is **convergent** if it is both confluent and terminating.

Theorem 1.3.8. *If an ARS $\mathcal{A}$ is convergent, then for any $a, b \in A$, $a \overset{*}{\leftrightarrow}_A b$ is decidable.*

Proof. The main point of this argument is that convergent reductions yield unique normal forms. Since $\to_A$ is terminating, all $a \in A$ have at least one normal form. For $a \in A$, let $n_1 \overset{*}{\leftarrow} a \overset{*}{\to} n_2$ be normal forms. Then there exists n such that $n_1 \overset{*}{\to} n \overset{*}{\leftarrow} n_2$, but since n_1 and n_2 are normal, $n_1 = n = n_2$. Then, the decision of $a \overset{*}{\leftrightarrow} b$ reduces to computing the unique normal forms of a and b and checking their equality. □

In certain cases, notably for terminating reductions, confluence can be reduced to the (often decidable) notion of local confluence.

Definition 1.3.9. A reduction relation $\to$ is said to be **locally confluent** if

$$\forall a, b, c \in A.(b \leftarrow a \to c \Rightarrow \exists d.(b \overset{*}{\to} d \overset{*}{\leftarrow} c))$$

Note that the elements b and c are only one rewrite step away, rather than possibly many.

Lemma 1.3.10. *(Newman) If $\to$ is locally confluent and terminating, it is confluent.*

1.3.3 Reduction Modulo an Equational Theory

For an ARS with domain A, an equational theory is a set of identities E on elements of A. Let $\equiv_E$ be the equivalence relation generated by E.

Often it is useful to treat true reductions (e.g. term substitution, evaluation of arithmetic operators, etc.) separately from structural congruences inherent in a system, such as association and commutation. Toward that end, we define reduction systems modulo an equational theory, as in [17].

Definition 1.3.11. For a reduction relation $\to_R$ and an equivalence relation $\equiv_E$, we define $\to_{R'} := \to_R / \equiv_E$ as follows, for all $a, b \in A$:

$$a \to_R' b \quad \iff \quad \exists a', b'.\ (a \equiv_E a') \wedge (b \equiv_E b') \wedge (a' \to_R b')$$

Often equivalence classes can be handled efficiently by building congruences into the data structure. For example, one can treat the arguments of commutative functions as multi-sets rather than lists. In our case, we shall subsume commutation by using general digraphs rather than digraphs where vertices have ordered edges.

1.3.4 Critical Pairs

A common means of proving local confluence is the study of critical pairs. A critical pair occurs when the LHS of two rewrite rules overlaps in a non-trivial way. In the case of terms, this occurs for two rewrite rules $l_1 \to r_1$ and $l_2 \to r_2$ when an instance of l_1 is a sub-term of an instance of l_2 (or vice-versa). By showing all terms containing critical pairs are joinable, one can prove local confluence. A notion of critical pairs as it applies to graphs will be introduced later in this chapter.

1.3.5 Completion

When a rewrite system is found to be non-confluent, it can often be "repaired" by adding **completions**.

If a rewrite system R has some non-joinable elements $b, c \in A$ such that $b \overset{*}{\leftarrow} a \overset{*}{\rightarrow} c$, one can define a rewrite system R' where b and c are joinable as $R' := R \cup \{b \to c\}$. Since we already have that $b \overset{*}{\leftrightarrow}_R c$, we have $\overset{*}{\leftrightarrow}_R = \overset{*}{\leftrightarrow}_{R'}$.

Often completions are chosen based on critical pair analysis. A simple completion algorithm would be to compute a critical pair $t_1 \leftarrow s \rightarrow t_2$ of a rewrite system R, then reduce t_1 and t_2 to normal forms $\widehat{t_1}$ and $\widehat{t_2}$. If $\widehat{t_1} = \widehat{t_2}$, continue to the next critical pair. If $\widehat{t_1} \neq \widehat{t_2}$, admit a rule $\widehat{t_1} \to \widehat{t_2}$ into R and start over. When all critical pairs of R are joinable, we have a confluent system. [3]

1.4 Graphs

Definition 1.4.1. Let a digraph G be a quintuple:

$$(V_G,\, E_G,\, s_G,\, d_G,\, \tau_G)$$

where V_G is a set of vertices, and E_G is a set of edges. The two functions $s_G : E_G \to V_G + \{0\}$ and $d_G : E_G \to V_G + \{0\}$ assign a source and a destination to an edge such that for all $e \in E_G$, $s_G(e) = 0 \Leftrightarrow d_G(e) = 0$. $\tau_G : V_G \to T$ is a typing function for a set of types T.

In the case that $s_G(e) = d_G(e) = 0$, e is called an empty edge, which is represented as a single edge connected to itself.

Let out_G be the inverse image of s_G and in_G be the inverse image of d_G. These are respectively the edges leaving from and entering a given vertex. For a graph G, and a type $t \in T$, let $t[V_G] =$

$\{v \in V_G : \tau_G(v) = t\}$. Also, for a set of types T, let

$$T[V_G] = \bigcup\{t[V_G] : t \in T\}.$$

A digraph homomorphism $f : G \to H$ is a pair of maps $f_v : V_G + \{0\} \to V_H + \{0\}$ and $f_e : E_G \to E_H$ that preserves the graph structure. That is to say, the following diagrams commute:

$$\begin{tikzcd} E_G \arrow[r, "s_G"] \arrow[d, "f_e"'] & V_G + \{0\} \arrow[d, "f_v"] \\ E_H \arrow[r, "s_H"'] & V_H + \{0\} \end{tikzcd} \qquad \begin{tikzcd} E_G \arrow[r, "d_G"] \arrow[d, "f_e"'] & V_G + \{0\} \arrow[d, "f_v"] \\ E_H \arrow[r, "d_H"'] & V_H + \{0\} \end{tikzcd}$$

Also, f_v must have the property that $f_v(0) = 0$. Note that a general digraph homomorphism does not necessarily preserve types. See the definition of type-soundness below.

Definition 1.4.2. A digraph homomorphism f is **strict** for a set of types $T' \subseteq T$ if

- f_e is injective,
- f_v restricted to $T'[V_G]$ is injective and
- for all $v \in T'[V_G], e \in E_G$,

$$e \in in_G(v) \Rightarrow f_e(e) \in in_H(f_v(v)) \text{ and } e \in out_G(v) \Rightarrow f_e(e) \in out_H(f_v(v)).$$

Definition 1.4.3. A digraph homomorphism is **type-sound** for a set of types $T' \subseteq T$ if for all $v \in T'[V_G]$, we have $\tau_G(v) = \tau_H(f_v(v))$.

Remark 1.4.4. If f is strict and type-sound, then the image of $f_v|_{T'[V_G]}$ is contained in $T'[V_H]$ and the following diagrams commute:

$$\begin{tikzcd} T'[V_G] \arrow[r, "in_G"] \arrow[d, "f_v|_{T'[V_G]}"'] & \mathcal{P}(E_G) \arrow[d, "\mathcal{P}(f_e)"] \\ T'[V_H] \arrow[r, "in_H"'] & \mathcal{P}(E_H) \end{tikzcd} \qquad \begin{tikzcd} T'[V_G] \arrow[r, "out_G"] \arrow[d, "f_v|_{T'[V_G]}"'] & \mathcal{P}(E_G) \arrow[d, "\mathcal{P}(f_e)"] \\ T'[V_H] \arrow[r, "out_H"'] & \mathcal{P}(E_H) \end{tikzcd}$$

In some cases, it is convenient to define undirected graphs as equivalence classes of directed graphs. For graphs G and H, let $G \equiv_D H$ iff H can be obtained from G by reversing the direction of zero or more edges. $\equiv_D$ clearly forms an equivalence relation, so for a digraph G, let $[G]_{\equiv_D}$ be the equivalence class under $\equiv_D$. We can treat $[G]_{\equiv_D}$ as an undirected graph.

1.4.1 Graph Matching and Rewriting

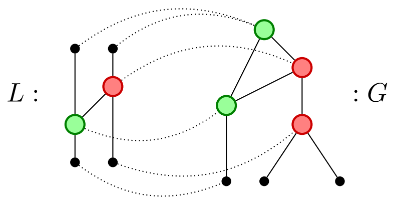
Assume that any set of types includes a special type $\beta \in T$, labelling **boundary vertices**. The set of types $I = T \backslash \{\beta\}$ labels **internal vertices**. The intuition is that if graphs represent some computation, boundary vertices are the inputs and outputs of that computation. We also now put the following restriction on boundary vertices:

$$\forall b \in \beta[V_G].|\mathit{in}_G(b)| + |\mathit{out}_G(b)| = 1$$

Since the set $\mathit{in}_G(b) \cup \mathit{out}_G(b)$ is a singleton for all b, we define the **boundary edge mapping** $be : \beta[V_G] \to E_G$, which maps b to the unique edge $e \in \mathit{in}_G(b) \cup \mathit{out}_G(b)$. If e is in $\mathit{out}_G(b)$ then b is called an **input**. Otherwise, b is called an **output**.

Definition 1.4.5. A digraph **matching** $m : G \to H$ is digraph homomorphism that is strict and type-sound on the set of internal types I.

In the following example, "red" and "green" are internal types and black dots are boundary vertices. A matching $m : L \to G$ is shown with dotted lines.



(1.2)

Definition 1.4.6. A **graph rewrite rule** is a pair of graphs L and R such that $V_L \cap V_R = \beta[V_L] = \beta[V_R]$.[2]

For a graph G, a rule $L \to R$, and a matching $m : L \to R$, we can define a new graph H as follows. Assume the graph G has distinct vertices and edges from L and R.

- $V_H = V_G \backslash m_v(I[V_L]) + I[V_R]$
- $E_H = E_G \backslash m_e(E_L) + m_e(E_R)$
- $s_H(e) = \begin{cases} s_G(e) & \textit{if } e \in E_G \\ m_v(s_R(e)) & \textit{if } s_R(e) \in \beta[V_R] \\ s_R(e) & \textit{otherwise} \end{cases}$

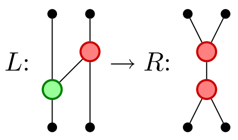
[2]More generally, we could define an isomorphism $\rho : \beta[V_L] \cong \beta[V_R]$. For simplicity, we are assuming $\rho = id_{\beta[V_L]}$.

- $d_H(e) = \begin{cases} d_G(e) & \textit{if } e \in E_G \\ m_v(d_R(e)) & \textit{if } d_R(e) \in \beta[V_R] \\ d_R(e) & \textit{otherwise} \end{cases}$

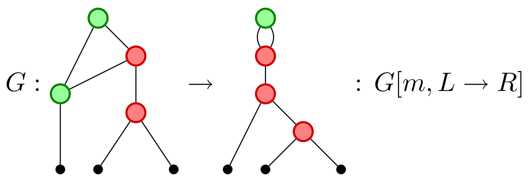
- $\tau_H(v) = \begin{cases} \tau_G(v) & \textit{if } v \in V_G \\ \tau_R(v) & \textit{otherwise} \end{cases}$

The mappings s_H and d_H are well-defined because $e \in E_H$ is in the domain of s_R (or d_R) precisely when it is not in E_G. When H is a rewrite of G, using a rewrite rule $L \to R$ and a matching $m : L \to G$, we write $H = G[m, L \to R]$.

Consider this rewrite rule:



The matching $m : L \to G$ from figure (1.2) yields the following graph rewrite:



Ideally, rewriting in this manner should be compositional. Rewriting on a sub-graph (i.e. a graph that has a matching on G) should be consistent with performing the same rewrite on G itself.

Lemma 1.4.7. *If $f : X \to Y$ and $g : Y \to Z$ are matchings, then $g \circ f$ is a matching.*

Proof. f_e and g_e are injective, so $g_e \circ f_e$ is injective. Type-soundness is also preserved in the composition, so the image of $f_v|_{T'[V_X]}$ is contained in $T'[V_Y]$. So, $(g_v \circ f_v)|_{T'[V_X]} = g_v|_{T'[V_Y]} \circ f_v|_{T'[V_X]}$, which is injective. Strictness is preserved because the following diagrams commute:

$$\begin{tikzcd}
I[V_X] \arrow[r, "in_X"] \arrow[d, "f_v"'] & \mathcal{P}(E_X) \arrow[d, "\mathcal{P}(f_e)"] \\
I[V_Y] \arrow[r, "in_Y"] \arrow[d, "g_v"'] & \mathcal{P}(E_Y) \arrow[d, "\mathcal{P}(g_e)"] \\
I[V_Z] \arrow[r, "in_Z"] & \mathcal{P}(E_Z)
\end{tikzcd}
\qquad
\begin{tikzcd}
I[V_X] \arrow[r, "out_X"] \arrow[d, "f_v"'] & \mathcal{P}(E_X) \arrow[d, "\mathcal{P}(f_e)"] \\
I[V_Y] \arrow[r, "out_Y"] \arrow[d, "g_v"'] & \mathcal{P}(E_Y) \arrow[d, "\mathcal{P}(g_e)"] \\
I[V_Z] \arrow[r, "out_Z"] & \mathcal{P}(E_Z)
\end{tikzcd}$$

□

Theorem 1.4.8. *(Composition) For graphs G and H, a rewrite rule $L \to R$, and matchings $p : H \to G$ and $m : L \to H$, then:*

$$G[p, H \to H[m, L \to R]] = G[p \circ m, L \to R]$$

Proof. Let G_1 be the LHS and G_2 be the RHS. We represent the edges of G is a disjoint union $E_{G'} + E_{H'} + E_{L'}$, where $E_{L'} = p \circ m(E_L)$, $E_{H'} + E_{L'} = p(E_H)$ and $E_{G'}$ is the set of remaining edges. Then:

$$\begin{aligned} E_{G_1} &= E_{G'} + E_{H[m,L\to R]} \\ &= E_{G'} + (E_{H'} + E_R) \\ &= (E_{G'} + E_{H'}) + E_R \\ &= E_{G_2} \end{aligned}$$

Similarly, $V_{G_1} = V_{G_2}$, as both sides are essentially replacing the non-boundary vertices of L with the non-boundary vertices of R. Now let e be an edge in E_{G_1} (or E_{G_2}), if $e \in E_{G'}$ or $e \in E_{H'}$, then it is fixed by either rewrite, so $e \in E_G$. Therefore $s_{G_1}(e) = s_G(e) = s_{G_2}(e)$.

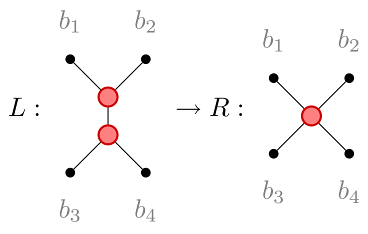
Otherwise, $e \in E_R$. If $s_R(e)$ is not a boundary vertex in R, then $s_{H[m,L\to R]} = s_R(e)$. Also, by type-soundness of m it cannot be a boundary vertex in $H[m, L \to R]$, so $s_{G_1} = s_{H[m,L\to R]} = s_R(e) = s_{G_2}(e)$.

If $s_R(e)$ is a boundary vertex in R, then it is a boundary vertex in L, so $m_v(s_R(e)) \in H[m, L \to R]$. This yields two cases. In the case that $m_v(s_R(e))$ is a boundary vertex in $H[m, L \to R]$, $s_{G_1}(e) = p_v(s_{H[m,L\to R]}(e)) = p_v(m_v(s_R(e))) = s_{G_2}(e)$. In the case that $m_v(s_R(e))$ is not a boundary vertex in $H[m, L \to R]$, then $s_{G_1}(e) = m_v(s_R(e))$. But, since it is a boundary vertex in R, it is fixed by the rewrite $G[p, H \to H[m, L \to R]]$. Any vertex v that is fixed by a rewrite on p has the property that $p_v(v) = v$. For the final case,

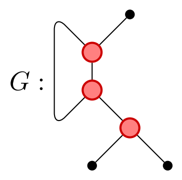
$$s_{G_1}(e) = m_v(s_R(e)) = p_v(m_v(s_R(e))) = s_{G_2}(e),$$

and similarly, $d_{G_1} = d_{G_2}$. □

This gives us almost the semantics we are looking for. However, using rewrite rules as they are given is too restrictive in the case of graphs that loop back on themselves. The problem lies in the strictness condition on m. Consider the following rewrite rule:



We would like to, for instance, be able to apply $L \to R$ to this graph:



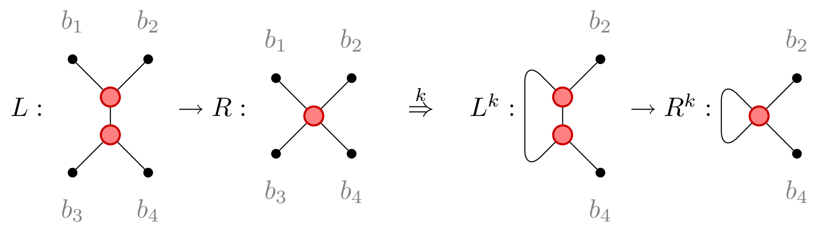
The strictness condition requires that any matching must be injective on edges. Therefore $L \to R$ has no matching on G. To correct this, we introduce the concept of a closure.

Definition 1.4.9. Let $O \subseteq \beta[V_H]$ be the set of outputs of H. A closure, $k \subseteq O \times \beta[V_H] \backslash O$ on a graph H is a set of ordered pairs where each $b \in \beta[V_H]$ occurs at most once. We can define H^k as the graph which connects each of the boundary vertex pairs listed in k. More specifically, we construct H^k as follows. Let B_k be the set of boundary vertices k (i.e. $\pi_1(k) \cup \pi_2(k)$).[3]

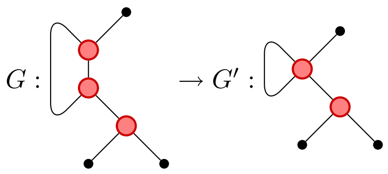
- $V_{H^k} = V_H \backslash B_k$
- $E_{H^k} = E_H \backslash be(B_k) \cup k$
- $s_{H^k}(e) = \begin{cases} s_H(e) & \textit{if } e \in E_H \\ s_H(be(\pi_1(e)) & \textit{if } e \in k \ \wedge \ s_H(be(\pi_1(e))) \in V_{H^k} \\ 0 & \textit{otherwise} \end{cases}$
- $d_{H^k}(e) = \begin{cases} d_H(e) & \textit{if } e \in E_H \\ d_H(be(\pi_2(e)) & \textit{if } e \in k \ \wedge \ d_H(be(\pi_2(e))) \in V_{H^k} \\ 0 & \textit{otherwise} \end{cases}$

The third cases in s and d are to ensure that the result of performing a closure on two connected boundary vertices is the empty edge. We can take the closure of a rewrite rule by applying the same k to both sides. Going back to the example from before, we could apply $k = \{(b_3, b_1)\}$ to obtain:

[3]Where π_1 and π_2 are the Cartesian projections, and $be : \beta[V_H] \to E_H$ is the boundary edge mapping defined at the beginning of this section. Note also that $s_H(be(\pi_1(e))$ and $d_H(be(\pi_2(e))$ are the vertices that were previously connected to the deleted boundary vertex.



Which does have a matching on G, yielding the graph rewrite:[4]



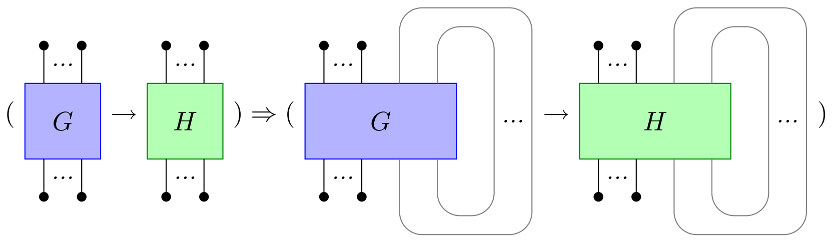
For S, a collection of graph rewrite rules, we define $\to_S$ as follows. For two graphs G and H, $G \to_S H$ if there exists $L \to R \in S$, a closure k, and a matching $m : L^k \to G$ such that $H = G[m, L^k \to R^k]$. We can define a rewrite system on undirected graphs as follows:

$$[G]_{\equiv_D} \to_S [H]_{\equiv_D} \iff \exists G \in [G]_{\equiv_D}, H \in [H]_{\equiv_D}.(G \to_S H)$$

Remark 1.4.10. For the set of finite typed digraphs $\mathcal{G}$ and a set of graph rewrite rules S, $(\mathcal{G}, \to_S)$ forms an abstract reduction system.

Proposition 1.4.11. *For disjoint closures h and k, we have $(G^h)^k = G^{(h\cup k)}$.*

Lemma 1.4.12. *(Closure) For a graph rewrite system S if we have $G \to_S H$, then for any closure k on G, we have $G^k \to_S H^k$. Graphically:*



Proof. If $G \to_S H$, then there must be a closure h of a rule $L \to R$ that has a match m on G. Let k_0 a one-element closure $\{(b_1, b_2)\}$. If neither b_1 nor b_2 is in the image of m, then m is still a matching for L^h.

If b_1 is in the image of m and b_2 is not, then let e be the edge where $s_G(e) = b_2$. For $b \in \beta[V_{L^h}]$ where $m(b) = b_1$, there is a matching $m' : L^h \to G^{k_0}$, where $m'_v(b) = d_G(e)$. For the edge e' where $d_{L^h}(e') = b$, $m'_e(e') = e^*$ where e^* is the new edge created by the closure. Thus m' is a homomorphism, and because b is a boundary vertex, strictness and type-soundness on $T\backslash\{\beta\}$ is

[4]The soundness of such a rewrite can be derived from compact closure. See theorem 2.2.1.

preserved. Rewriting on the new matching m' only varies from rewriting on m by its behaviour on the closure edge. A symmetric argument applies for b_1 not in the image of m and b_2 in the image of m.

If b_1 and b_2 are in the image of m, then $L^{(h \cup k_0)}$ clearly has a matching on G^{k_0}, and this matching also only varies from m in its behaviour on the closure edge. We can apply proposition 1.4.11 to generalise to any finite closure $k = k_0 \cup k_1 \cup \ldots \cup k_n$. □

1.4.2 Critical Pair Enumeration

Definition 1.4.13. Let graphs G and H have matchings u^G and u^H into a graph U. We call the triple (U, u^G, u^H) a unification if these conditions are satisfied:

- $V_U = u^G_v(V_G) \cup u^H_v(V_H)$
- $E_U = u^G_e(E_G) \cup u^H_e(E_H)$
- $I[V_U] = u^G_v(I[V_G]) \cup u^H_v(I[V_H])$

Definition 1.4.14. For a unification (U, u^G, u^H), an **overlap** u is a pair of partial functions $u_v : I[V_G] \to I[V_H]$ and $u_e : E_G \to E_H$, defined as follows:

- $u_v :: \{v_1 \mapsto v_2 : u^G_v(v_1) = u^H_v(v_2)\}$
- $u_e :: \{e_1 \mapsto e_2 : u^G_e(e_1) = u^H_e(e_2)\}$

Proposition 1.4.15. *u_v and u_e are well-defined and injective.*

Proof. Let $u_v :: \{v_1 \mapsto v_2, v_1 \mapsto v_3\}$. Then $u^H_v(v_2) = u^G_v(v_1) = u^H_v(v_3)$. Since $u^H_v|_{I[V_H]}$ is injective, $v_2 = v_3$. Let $u_v :: \{v_1 \mapsto v_3, v_2 \mapsto v_3\}$. Then $u^G_v(v_1) = u^H_v(v_3) = u^G_v(v_2)$. Since $u^G_v|_{I[V_G]}$ is injective, $v_1 = v_2$. Similarly, these properties for u_e follow from u^G_e and u^H_e being injective. □

Proposition 1.4.16. *u uniquely defines a unification (U, u^G, u^H). In other words, for any unification (U', m^G, m^H) with overlap $m = u$, there exists a type-sound graph isomorphism $\pi : U \cong U'$ such that the following diagram commutes:*

$$\begin{tikzcd}
G \arrow[r, "u^G"] \arrow[dr, "m^G"'] & U \arrow[d, equal, "\pi"] & H \arrow[l, "u^H"'] \arrow[dl, "m^H"] \\
 & U' &
\end{tikzcd}$$

Proof. Define π_e as $\{u_e^G(e) \mapsto m_e^G(e) : e \in E_G\} \cup \{u_e^H(e) \mapsto m_e^H(e) : e \in E_H\}$. This is well defined and injective because $\forall e.(u_e^G(e) = u_e^H(e) \Leftrightarrow m_e^G(e) = m_e^H(e))$. It is surjective because U is a unification. Define π on $I[V_U]$ similarly. This is a bijection from $I[V_U]$ to $I[V_{U'}]$ and furthermore is type-sound, as u^G, u^H, m^G, and m^H are all type-sound. For $b \in \beta[V_U]$, there exists exactly one edge e in $in_U(v) \cup out_U(v)$. If $e \in in_U(v)$, then $b = d_U(e)$. Let $\pi_v(b) = d_{U'}(\pi_e(e))$. If $e \in out_U(v)$, then $b = s_U(e)$. Let $\pi_v(b) = s_{U'}(\pi_e(e))$. This defines a bijection from $\beta[V_U]$ to $\beta[V_{U'}]$, because we can define the inverse π_v^{-1} on boundary vertices using π_e^{-1}.

For $e \in E_U$, if $s_U(e)$ is a boundary vertex, then $\pi_v(s_U(e)) = s_{U'}(\pi_e(e))$ by construction. If it is not a boundary vertex, recall that U is a unification, so $e \in u_e^G(E_G) \cup u_e^H(E_H)$. Assume it is in $u_e^G(E_G)$ and take $e' \in E_G$ such that $u_e^G(e') = e$. Then, by definition, $m_e^G(e') = \pi_e(e)$ and $m_v^G(s_G(e')) = \pi_v(u_v^G(s_G(e')))$, so

$$s_{U'}(\pi_e(e)) = s_{U'}(m_e^G(e')) = m_v^G(s_G(e')) = \pi_v(u_v^G(s_G(e'))) = \pi_v(s_U(u_e^G(e')) = \pi_v(s_U(e))$$

by definition. The same is true if $e \in u_e^H(E_H)$. Similarly, $d_{U'}(\pi_e(e)) = \pi_v(d_U(e))$. Therefore π is in fact an isomorphism. By construction, the following diagrams commute:

$$\begin{tikzcd}
E_G \arrow[r, "u_e^G"] \arrow[dr, "m_e^G"'] & E_U \arrow[d, equal, "\pi_e"] & E_H \arrow[l, "u_e^H"'] \arrow[dl, "m_e^H"] \\
& E_{U'} &
\end{tikzcd}
\qquad
\begin{tikzcd}
I[V_G] \arrow[r, "u_v^G"] \arrow[dr, "m_v^G"'] & I[V_U] \arrow[d, equal, "\pi_v"] & I[V_H] \arrow[l, "u_v^H"'] \arrow[dl, "m_v^H"] \\
& I[V_{U'}] &
\end{tikzcd}$$

The edge diagram implies the same for boundary vertices, so the diagram stated in the proposition commutes. □

As overlaps define unique unifications, we often write $G +_u H$ for the unification of G and H with overlap u.

Lemma 1.4.17. *For any graph G, if graphs H_1 and H_2 have matchings $m : H_1 \to G$ and $n : H_2 \to G$, then there exists an overlap $u : H_1 \to H_2$ such that $H_1 +_u H_2$ has a matching on G.*[5]

Proof. Let u be the overlap defined by m and n. For $v_1 \in H_1$, $v_2 \in H_2$, $(m(v_1) = n(v_2)) \Rightarrow \big(u^{H_1}(v_1) = u^{H_2}(v_2)\big)$, so a digraph homomorphism p that makes the following diagram commute exists and is unique.

[5]Alternatively, we could say $(H_1 +_u H_2, u^{H_1}, u^{H_2})$ is a push-out defined in a suitable category, of (O, i_1, i_2), where O is the shared portion of H_1 and H_2 and i_1, i_2 are the inclusion maps.

$$\begin{tikzcd}
H_1 \arrow[d, "u^{H_1}"'] \arrow[drr, "m"] & & \\
H_1 +_u H_2 \arrow[rr, "p"] & & G \\
H_2 \arrow[u, "u^{H_2}"] \arrow[urr, "n"'] & &
\end{tikzcd}$$

Strictness and type-soundness follow from the observation that every edge and internal vertex in $H_1 +_u H_2$ is in the image of u^{H_1} or u^{H_2}.

□

Definition 1.4.18. For rules $r_1 : L_1 \to R_1$ and $r_2 : L_2 \to R_2$ we define a **critical pair** (G_1, G_2) for pair of closures k_1, k_2 and an overlap $u : L_1^{k_1} \to L_2^{k_2}$. Let G_i be the result of rewriting $L_1^{k_1} +_u L_2^{k_2}$, using the matching $u^{L_i^{k_i}}$ for the rule $r_i^{k_i}$.

Lemma 1.4.19. *(Critical pair) If for all critical pairs (G_1, G_2) in a rewrite system S, there exists some H such that $G_1 \xrightarrow{*}_S H \xleftarrow{*}_S G_2$, then S is locally confluent.*

Proof. Let G, H_1, H_2 be graphs such that $H_1 \leftarrow_{r_1} G \rightarrow_{r_2} H_2$, for rewrite rules r_1 and r_2. Let each $r_i : L_i \to R_i$ have an associated closure k_i and matching m_i on to G. Let u be as defined in the proof of 1.4.17 for matchings m_1 and m_2. $L_1^{k_1} +_u L_2^{k_2}$ has a matching p on G, as above, and generates a critical pair (C_1, C_2). Since $m_i = p \circ u^{H_i}$, and $C_i = (L_1^{k_1} +_u L_2^{k_2})[u^{H_i}, L_i^{k_i} \to R_i^{k_i}]$, so by theorem 1.4.8 we now have:

$$G[p, L_1^{k_1} +_u L_2^{k_2} \to C_i] = G[m_i, L_i^{k_i} \to R_i^{k_i}] = H_i$$

Also, C_1 and C_2 are joinable to some graph U, so the result of replacing C_i with U yields a graph G' with the property that $H_1 \xrightarrow{*} G' \xleftarrow{*} H_2$. □

The number of critical pairs that need to be checked can be reduced by identifying closures that interact trivially with the formation of critical pairs.

Proposition 1.4.20. *For a critical pair (C_1, C_2), if there exists a closure k and a joinable critical pair (D_1, D_2) such that $C_1 = D_1^k$ and $C_2 = D_2^k$ then (C_1, C_2) is joinable.*

Proof. Follows immediately from lemma 1.4.12. □

Proposition 1.4.21. *For a pair of rewrite rules $r_1 : L_1 \to R_1$, $r_2 : L_2 \to R_2$, if L_1 and L_2 are acyclic, then for any closures k_1, k_2, the critical pairs of $(r_1^{k_1}, r_2^{k_2})$ are all of the form (D_1^k, D_2^k) for a closure k and a critical pair (D_1, D_2) of (r_1, r_2).*

For a pair of rewrite rules $r_1 : L_1 \to R_1$, $r_2 : L_2 \to R_2$, each critical pair is determined uniquely by the chosen overlap. Therefore, we can find every possible critical pair by looking at the set of all partial graph homomorphisms $u : L_1 \to L_2$ and consider only those that yield valid overlaps. If L_1 and L_2 are acyclic and all of the critical pairs of (r_1, r_2) are joinable, we are done. Otherwise, we examine the critical pairs of $(r_1^{k_1}, r_2^{k_2})$ for all closures k_1, k_2.

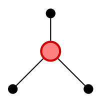
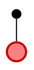
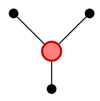
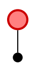
Chapter 2

Rewriting with a Single Classical Structure

In this chapter, we shall introduce a rewrite system Q_1, which permits rewriting with a single classical structure modulo association and frobenius. Q_1 acts on a set of graphs with types $T = \{\beta, \mathsf{r}\}$. We will then define a type of graph called a spider and show that Q_1 permits writing any arbitrary graph of a single classical structure to a spider. Using this, we show the convergence of Q_1.

2.1 Classical Structures as Graph Components

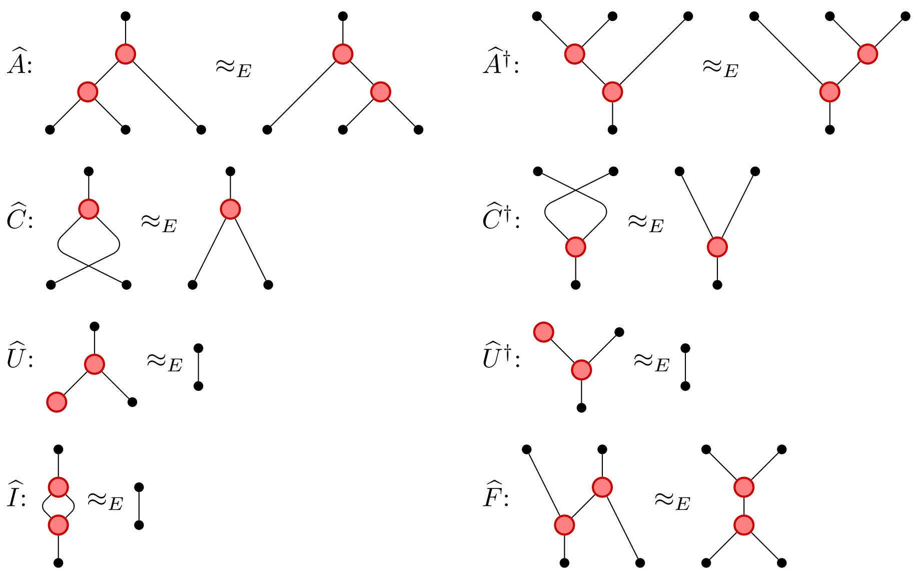
We can express a classical structure $(\delta_Z, \epsilon_Z, A)$ as:

$\delta_Z :=$  $\epsilon_Z :=$  $\delta_Z^\dagger :=$  $\epsilon_Z^\dagger :=$ 

Since δ_Z is cocommutative and $\delta_Z^\dagger$ is commutative, general graphs will suffice for describing morphisms composed of these components. We have refrained here from labelling the vertex itself because the intended component is clear from its arity. Graphs in this section are to be read top-to-bottom, so edges connected to the tops of vertices are understood to be inputs and edges connected to the bottom are outputs.

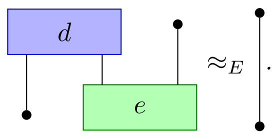
2.2 An Equational System

For a classical structure $(A, \delta_Z : A \to A \otimes A, \epsilon_Z : A \to I)$ in a †-SMC, we can define the following equational system:

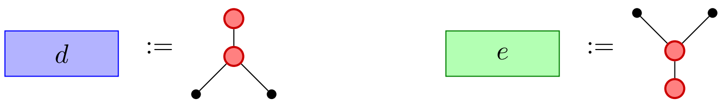


$\widehat{A}$, $\widehat{C}$[1], and $\widehat{U}$ arise from $(A, \delta_Z, \epsilon_Z)$ forming a cocommutative comonoid. The "daggered" versions of these rules arise from the consistency of $(-)^\dagger$ with the comonoidal structure. $\widehat{I}$ is self-dual with respect to the dagger. The dual of $\widehat{F}$ can be derived from $\widehat{F}$ and commutation.

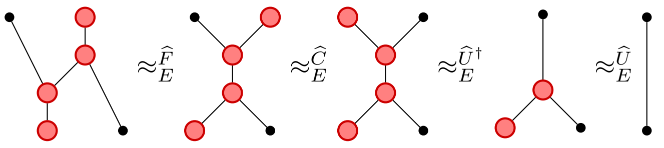
Theorem 2.2.1. *The graph components $\delta_Z, \epsilon_Z, \delta_Z^\dagger, \epsilon_Z^\dagger$ and the equational system $\approx_E$ yield a compact closed structure. i.e. there exist graphs d and e such that:*



Proof. In theorem 1.2.3, we gave this result in the categorical semantics. We now show it graphically by letting

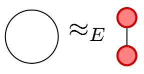


Then, the equation follows from frobenius and unit rules:

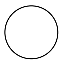
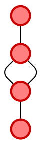
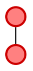

$\approx_E^{\widehat{F}}$
$\approx_E^{\widehat{C}}$
$\approx_E^{\widehat{U}^\dagger}$
$\approx_E^{\widehat{U}}$

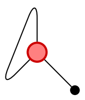
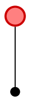
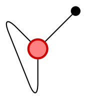
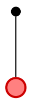
□

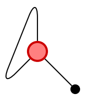
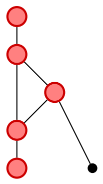
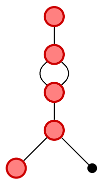
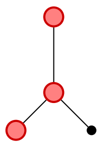
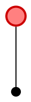
Corollary 2.2.2.

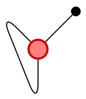
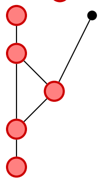
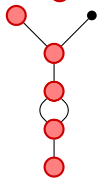
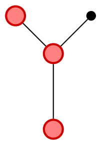
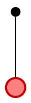


[1] $\widehat{C}$ and $\widehat{C}^\dagger$ are identities in the case of general graphs, where vertices do not have an edge ordering. They are included here merely for the sake of completeness.

Proof.  $\approx_E^{CC}$  $\approx_E^{\widehat{I}}$  □

Corollary 2.2.3.  $\approx_E$  *and*  $\approx_E$ 

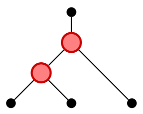
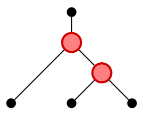
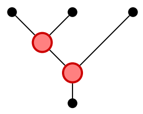
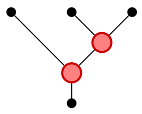
Proof.  $\approx_E^{CC}$  $\approx_E^{\widehat{F}}$  $\approx_E^{\widehat{I}}$  $\approx_E^{\widehat{U}}$ 

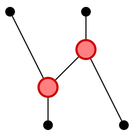
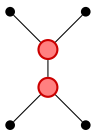
 $\approx_E^{CC}$  $\approx_E^{\widehat{F}}$  $\approx_E^{\widehat{I}}$  $\approx_E^{\widehat{U}^\dagger}$  □

2.3 The Rewrite System

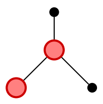
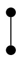
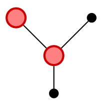
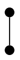
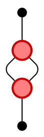
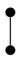
From the above equational system, we wish to derive a confluent rewrite system. It is useful to differentiate which equations denote actual reductions (i.e. they decrease the size of the graph) and which denote structural congruences (i.e. they translate into a graph that is "symmetric" in some sense). For the first, we choose $\widehat{I}$, $\widehat{U}$ and $\widehat{U}^\dagger$. For the latter, we choose $\widehat{A}$, $\widehat{C}$, and $\widehat{F}$. We will consider the congruence rules as an equational system and the reductive rules as a rewrite system modulo this equational system.

We begin by defining a smaller equational system $\approx_{AF}$:

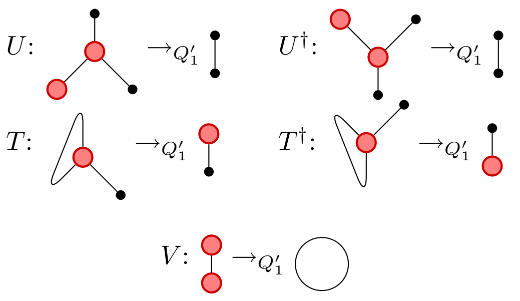
A:  $\approx_{AF}$  $A^\dagger$:  $\approx_{AF}$ 

F:  $\approx_{AF}$ 

We first attempt to form a rewrite system by directing all the remaining rules from bigger to smaller graphs.

U:  $\rightarrow_R$  $U^\dagger$:  $\rightarrow_R$  I:  $\rightarrow_R$ 

But we already know this doesn't decide E, because none of the identities in corollaries 2.2.2 and 2.2.3 are present as rewrites in R. Instead, let us examine the following system, using the derived rules from corollaries 2.2.2 and 2.2.3:



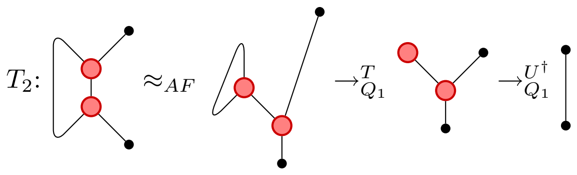
We define $\rightarrow_{Q_1}$ modulo $\approx_{AF}$, as in section 1.3.3. Let $\rightarrow_{Q_1} := \rightarrow_{Q'_1} / \approx_{AF}$. Note that this system omits an isometry rule. We shall see in the next section that this can now be recovered as a derived rule from T and $T^\dagger$.

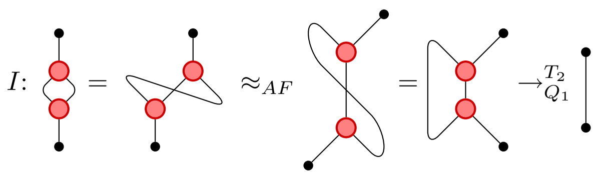
2.3.1 Soundness and Power of the Rewrite System

Proposition 2.3.1. $\overset{*}{\rightarrow}_{Q_1} \subseteq \approx_E$.

Proof. If we show each individual rewrite rule is derivable from the equational system, then by RST-closure of $\approx_E$, we have $\overset{*}{\rightarrow}_{Q_1} \subseteq \approx_E$. All rewrites besides the trace rules are simply directed versions of rules in E. For the trace rules, refer to 2.2.3 for the proof they are contained in E. ☐

At first glance, this rewrite system seems to have omitted isometry, but this rule is actually derivable from T and F.

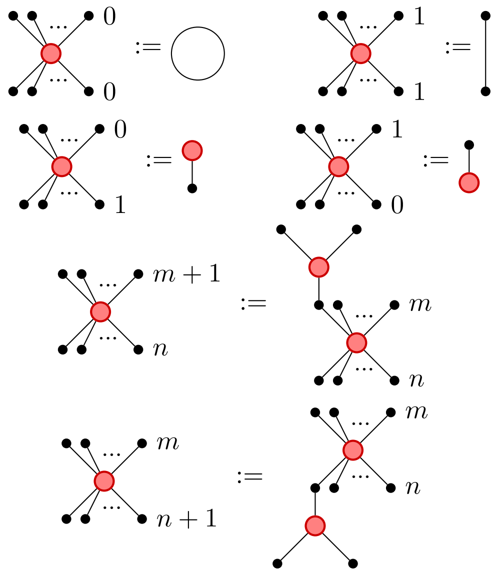




The derived rules I and T_2 will be used in later results.

2.4 Spiders

Spiders are a useful generalisation of one colour graphs. We define them with the following recursion equations:



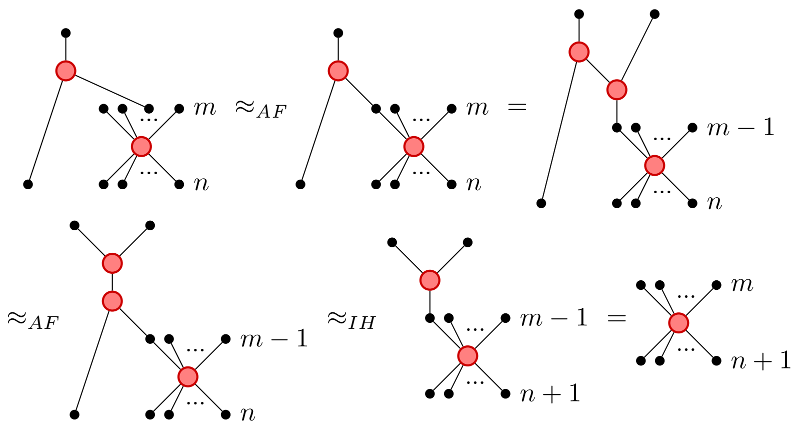
2.4.1 The Spider Theorem

Definition 2.4.1. A δ-tree is a (fully) acyclic graph containing only δ and $\delta^\dagger$.

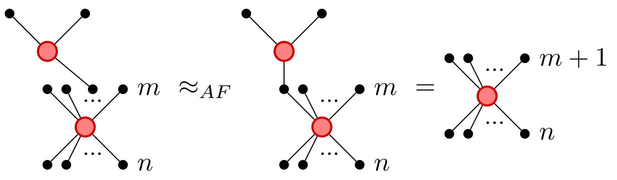
Proposition 2.4.2. *Any δ-tree with m inputs and n outputs is equivalent modulo $\approx_{AF}$ to a spider with m inputs and n outputs.*

Proof. δ-trees containing a single vertex are already spiders. Suppose for the sake of induction that all δ-trees with $\leq k$ vertices are equivalent to the corresponding spider. Consider all of the ways an additional δ are $\delta^\dagger$ can be connected.

Case I



Case II



Case III

Adjoint to Case I.

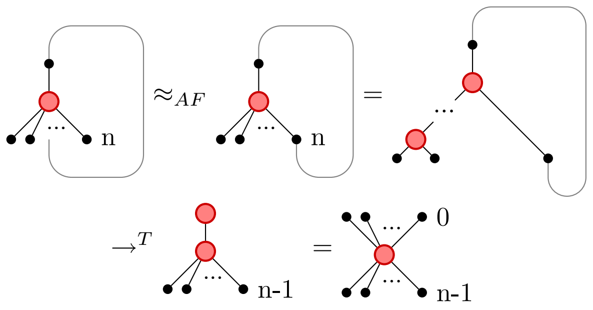
Case IV

Adjoint to Case II. ☐

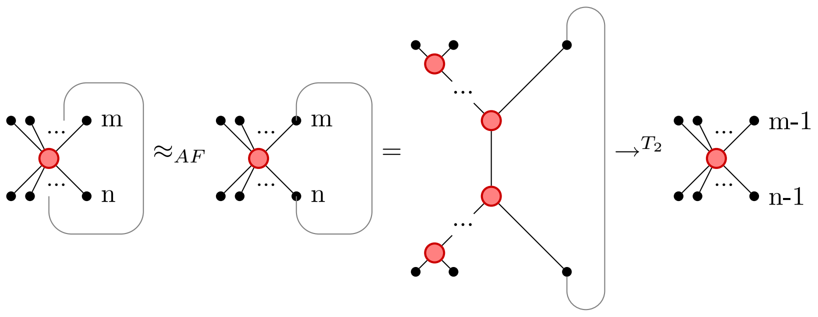
This proposition in conjunction with the following trace lemma will be adequate to prove the spider theorem.

Lemma 2.4.3. *(Trace) Any spider from $A^{\otimes m}$ to $A^{\otimes n}$ with k outputs connected to k inputs can be rewritten as a spider from $A^{\otimes(m-k)}$ to $A^{\otimes(n-k)}$.*

Proof. Consider the sub-graph containing a spider and just one trace. We divide into three cases, based on the values of m and n. If m or n is 0, there can be no traces. If $m = n = 1$ then tracing these together yields a single empty edge. If $m = 1$ and $n > 1$, we have:



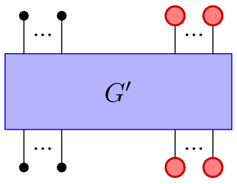
The dual proof holds for $m > 1, n = 1$. For $m > 1$ and $n > 1$ we have:



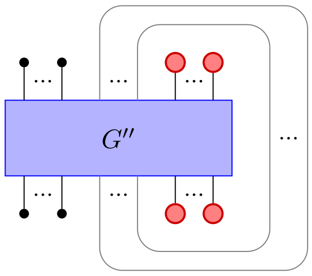
If we repeat this process k times, we get the required result. ☐

Theorem 2.4.4. *Any graph consisting of δ, ϵ, $\delta^\dagger$ and $\epsilon^\dagger$ can be rewritten to a spider.*

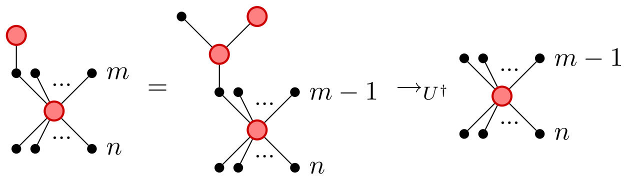
Proof. Consider an arbitrary graph G of δ, ϵ, $\delta^\dagger$ and $\epsilon^\dagger$ with m inputs and n outputs. Let x be the number of $\epsilon^\dagger$ vertices and y the number of ϵ vertices. We rewrite G as a graph G' from $A^{\otimes(m+x)}$ to $A^{\otimes(n+y)}$, with all ϵ and $\epsilon^\dagger$ on inputs or outputs:



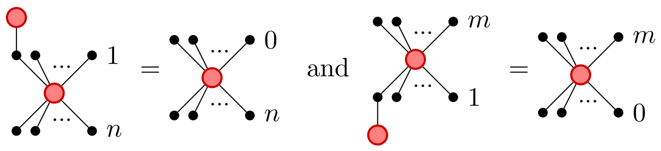
Now, let $G'' : A^{\otimes(m+x+k)} \to A^{\otimes(n+y+k)}$ be a spanning tree of G'. We can now express G' as G with all k edges in $G'\backslash G''$ drawn as external traces.



G'' is a δ-tree, so by proposition is equivalent to a spider. By the trace lemma, the traces disappear. If G'' has at least two inputs, we can eliminate an $\epsilon^\dagger$ as follows:



If $m = 0$, we can repeat this process $x - 1$ times to eliminate all but one $\epsilon^\dagger$ vertex. If $m > 0$, we can repeat this x times and eliminate all $\epsilon^\dagger$ vertices. We proceed similarly for ϵ vertices and outputs. Recall that:



So the end result is a spider from $A^{\otimes m}$ to $A^{\otimes n}$.

□

2.4.2 Confluence and Termination

Remark 2.4.5. As U, $U^\dagger$, T and $T^\dagger$ strictly decrease the number of δ and $\delta^\dagger$ vertices in the graph, this rewrite system is necessarily terminating.

Proposition 2.4.6. *R is confluent.*

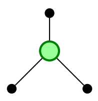
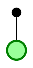
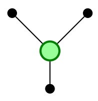
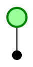
Proof. Let G be a graph with m inputs and n outputs. Let H_1 and H_2 be graphs such that $H_1 \overset{*}{\leftarrow} G \overset{*}{\rightarrow} H_2$. Graph rewrite systems must preserve the number of inputs and outputs to a graph. Therefore, H_1 and H_2 have m inputs and n outputs. By theorem 2.4.4, they both rewrite to the spider from $A^{\otimes m}$ to $A^{\otimes n}$. $\square$

R is confluent and terminating, so it is convergent. Spiders are acyclic and contain no (reducible) occurrences of ϵ or $\epsilon^\dagger$, so they are normal forms. Since R is convergent, they are the only normal forms.

Chapter 3

Complementary Classical Structures

In section 1.2, we introduced the concept of "complementary classical structures." We can represent such structures by admitting two colours of dots into our graph system. So for $(\delta_Z, \epsilon_Z, A)$, we maintain the red dots and additionally define:

$\delta_X :=$  $\epsilon_X :=$  $\delta_X^\dagger :=$  $\epsilon_X^\dagger :=$ 

The two colours are distinguished using types. The set of types for a two-colour graph is $T := \{\beta, \mathsf{r}, \mathsf{g}\}$. Also, due to the spider theorem and the self-duality of A it is not necessary to distinguish inputs from outputs in graph elements. For this reason, we will use undirected graphs from now on.

3.1 Graphs of Spiders

To begin, we will develop the mechanism for dealing with spiders as primitive graph elements. A graph of spiders is simply an arbitrary, undirected graph on the types $T = \{\beta, \mathsf{r}, \mathsf{g}\}$. We can use the results from the previous section if we interpret vertices of incidence 0 as the trivial graph, vertices of incidence 1 as ϵ, vertices of incidence 2 as identities, and vertices of incidence $n > 2$ as an appropriate δ-tree. To make the distinction with graph patterns below, we often call such a graph a **concrete graph of spiders**.

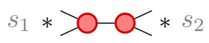
In order to match on spiders, we introduce the concept of **graph patterns**. Graph patterns are graphs over the set of types $T^* = \{\beta, \mathsf{r}, \mathsf{g}, \mathsf{r*}, \mathsf{g*}\}$. The types $\mathsf{r*}$ and $\mathsf{g*}$ are called **$*$-types**. For convenience, let $S \subset T^*$ be the set of $*$-types and $S[V_P]$ the set of **$*$-vertices** in a graph pattern P. We can define an instance of a graph pattern as follows.

Definition 3.1.1. Given a graph pattern P, $(B, \sigma : S[V_P] \to \mathcal{P}(B))$ defines an **instance** G of P if:

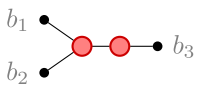
- G is a concrete graph of spiders
- σ is a partition of B
- $V_G = V_P \cup B$
- $E_G = E_P \cup \{(v, b) : b \in \sigma(v)\}$
- $\tau_G(v) = \begin{cases} \beta & \textit{if } v \in B \\ \mathsf{r} & \textit{if } \tau_P(v) = \mathsf{r*} \\ \mathsf{g} & \textit{if } \tau_P(v) = \mathsf{g*} \\ \tau_P(v) & \textit{otherwise} \end{cases}$

The intuition is that σ gives each $*$-vertex a set of concrete boundary vertices. By abuse of notation, we write G as σP.

Graphically, we can represent a graph pattern P as follows:



So, for $\sigma :: \{s_1 \mapsto \{b_1, b_2\}, s_2 \mapsto \{b_3\}\}$, we can construct an instance σP:



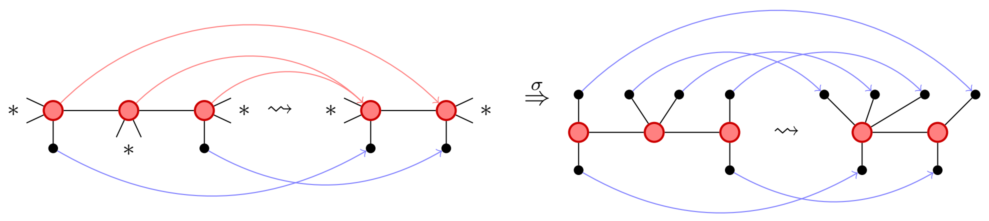
Definition 3.1.2. A **graph pattern rewrite rule** $(P \rightsquigarrow Q, \chi)$ consists of a pair of graph patters P, Q such that $P \to Q$ is a graph rewrite rule and $\chi : S[V_P] \to S[V_Q]$ is surjective.

Since χ is surjective, its inverse image $\chi^{-1} : S[V_Q] \to \mathcal{P}(S[V_P])$ is total, and furthermore defines a partition of $S[V_P]$. For an instance (B, σ), we can construct a new instance (B, σ_χ) by setting $\sigma_\chi(v) = \bigcup \sigma(\chi^{-1}(v))$ for all $v \in S[V_Q]$. Since χ^{-1} and the image mapping $\mathcal{P}(\sigma) : \mathcal{P}(S[V_P]) \to \mathcal{P}(B)$ are both partitions, σ_χ is a partition.

Proposition 3.1.3. *For a graph pattern rewrite rule $(P \rightsquigarrow Q, \chi)$, and an instance (B, σ) of P, (B, σ_χ) is an instance of Q and $\sigma P \to \sigma_\chi Q$ defines a graph rewrite rule.*

Proof. It suffices to show that $V_{\sigma P} \cap V_{\sigma_\chi Q} = \beta[V_{\sigma P}] = \beta[V_{\sigma_\chi Q}]$. Since $P \to Q$ is a rewrite rule, we know $V_P \cap V_Q = \beta[V_P] = \beta[V_Q]$. $V_{\sigma P} = V_P + B$, $V_{\sigma_\chi Q} = V_Q + B$. Since B is made up entirely of boundary vertices in σP and $\sigma_\chi Q$, the condition holds. □

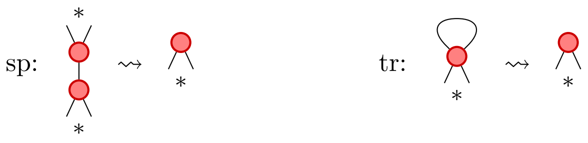
Below is a rewrite pattern and an instance, with χ marked in red and shared boundary vertices marked in blue:



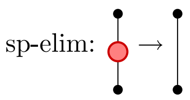
For a graph pattern rewrite system R, we define $\to_R$ as $G \to_R H$ if there exists a rule $(P \rightsquigarrow Q, \chi)$, and instance σ and a closure k (as defined in section 1.4.1) such that H can be formed by replacing $(\sigma P)^k$ with $(\sigma_\chi Q)^k$. This is a strict generalisation of a rewrite system, since $(P \rightsquigarrow Q, \emptyset)$ is the same as $P \to Q$, provided neither graph contains $*$-vertices. We often omit χ if it is clear from context.

3.2 The Spider Theorem with Graph Patterns

Since we can interpret vertices as δ-trees, the spider theorem justifies the following rewrites:



Since we are now treating spiders as primitives, we also need a rule to get rid of them. We do so by this generalisation of the (co)monoidal unit rewrite:



3.2.1 $*$-Rewriting

It would be useful to do graph rewriting with a graph pattern itself, rather than the set of all instances. To do this, we first need a relaxed version of matching for graph patterns. For convenience, we identify the set of $*$-types $S = \{\mathsf{r*}, \mathsf{g*}\}$, the set of non-$*$, or concrete types $C = \{\mathsf{r}, \mathsf{g}\}$, and the set of non-boundary, or internal types $I = C \cup S$. The full set is then $T^* = I \cup \{\beta\}$.

Definition 3.2.1. A graph homomorphism f is **$*$-type sound** if it is type-sound on C and for all v:

$$\tau(v) = \mathsf{r*} \quad \Rightarrow \quad \tau(f_v(v)) \in \{\mathsf{r}, \mathsf{r*}\}$$

$$\tau(v) = \mathsf{g*} \quad \Rightarrow \quad \tau(f_v(v)) \in \{\mathsf{g}, \mathsf{g*}\}$$

Definition 3.2.2. A graph homomorphism f is **$*$-strict** if it is strict on C, injective on I, and has the property that $f_v(\beta[V_P]) \cap f_v(S[V_P]) = \emptyset$.

Remark 3.2.3. Previously, strictness was enough to ensure that the LHS of a rewrite rule could not "hook back" onto itself, leading to undefined behaviour. Since we remove this constraint on S, we need to supply $m_v(\beta[V_P]) \cap m_v(S[V_P]) = \emptyset$ explicitly.

Definition 3.2.4. For graph patterns P and G, **$*$-matching** $m : P \to G$ is a graph homomorphism that is $*$-type sound and $*$-strict.

For a graph G and a pattern rewrite rule $p = (P \rightsquigarrow Q, \chi)$, and a $*$-matching $m : P \to G$, we can construct a new graph H as follows.

- $V_H = V_G \backslash m_v(I[V_P]) \cup I[V_Q]$
- $E_H = E_G \backslash m_e(E_P) \cup m_e(E_Q)$
- $s_H(e) = \begin{cases} s_G(e) & \textit{if } e \in E_G,\ s_G(e) \in V_G \\ s_Q(e) & \textit{if } e \in E_Q,\ s_Q(e) \in I[V_Q] \\ \chi(v) & \textit{if } e \in E_G,\ v \in S[V_P],\ s_G(e) = m_v(v) \\ m_v(s_Q(e)) & \textit{otherwise} \end{cases}$
- $d_H(e) =$ (as above, replacing "s" with "d")
- $\tau_H(v) = \begin{cases} \tau_G(v) & \textit{if } v \in V_G \\ \mathsf{r} & \textit{if } \tau_Q(v) = \mathsf{r*} \\ \mathsf{g} & \textit{if } \tau_Q(v) = \mathsf{g*} \\ \tau_Q(v) & \textit{otherwise} \end{cases}$

As with concrete rewriting, we can express H as $G[m, (P \rightsquigarrow Q, \chi)]$.

Remark 3.2.5. In the above definition, m is injective on $S[V_P]$, so $\chi(v)$ in the definitions of s_H and d_H is uniquely determined. To understand these functions, think of the four cases as follows:

(i) edges totally external to the rewrite

(ii) edges totally internal to the rewrite

(iii) edges left dangling by lack of strictness on $*$-vertices

(iv) boundary edges

By analogy to concrete rewrite rules, we can express H as $G[m, P \rightsquigarrow Q]$.

Proposition 3.2.6. *For graphs G, H, and a pattern rewrite rule $(P \rightsquigarrow Q, \chi)$*

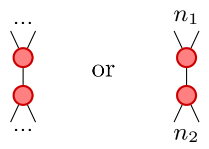
$$\exists \widehat{m}.\ G[\widehat{m}, (P \rightsquigarrow Q, \chi)] = H \qquad \Longleftrightarrow \qquad \exists m, \sigma.\ G[m, \sigma P \to \sigma_\chi Q] = H$$

where m is a matching, σ is an instance, and $\widehat{m}$ is a $*$-matching.

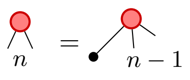
Proof. ($\Leftarrow$) Let $\widehat{m}$ be m restricted to P. $\widehat{m}$ is $*$-type-sound because m is type-sound, and it is strict on C because the only omitted edges are incident to $*$-vertices. For vertices and edges not incident to a $*$-vertex, both rewrites behave identically. For any $*$-vertex s, any vertex in H that was adjacent to $m(s)$ $(= \widehat{m}(s))$ in G will be adjacent to $\chi(s)$ in both $G[\widehat{m}, (P \rightsquigarrow Q, \chi)]$ and $G[m, \sigma P \to \sigma_\chi Q]$.

($\Rightarrow$) For each $s \in S[V_P]$, let $\sigma(s) = \{b_{s,1}, b_{s,2}, \ldots, b_{s,r}\}$, where $(b_{s,i})_i$ are new boundary vertices, and r is $(|\,in_G(\widehat{m}(s))| + |\,out_G(\widehat{m}(s))|) - (|\,in_P(s)| + |\,out_P(s)|)$. I.e. r is the number of *extra* edges needed to make s strict. $\widehat{m}$ then extends naturally to a matching to m. Since $\widehat{m}$ is the restriction of m to P, as above, the two rewrites are equivalent. $\square$

Often with $*$-rewriting, it is useful to reason about graphs with arbitrary boundary edges. We do this by labelling a vertex with a natural number or ellipses:



Since $*$-matchings need not be strict on $*$-vertices, the behaviour of $*$-rewriting on such graphs is well-defined. As one would expect, we require the following to hold:

For $n \geq 1$: 

Definition 3.2.7. For a graph pattern P, a **$*$-closure** is a set of ordered pairs $k \subseteq (S[V_P] \cup \beta[V_P]) \times \beta[V_P]$ where each boundary vertex occurs in k at most once. We define P^k as follows (for B_k the set of boundary vertices that occur in k):

- $V_{P^k} = V_P \backslash B_k$
- $E_{P^k} = E_P \backslash be(B_k) \cup k$
- $s_{P^k}(e) = \begin{cases} s_P(e) & \textit{if } e \in E_P \\ \pi_1(e) & \textit{if } e \in k \ \wedge \ \pi_1(e) \in S[V_P] \\ s_P(be(\pi_1(e)) & \textit{if } e \in k \ \wedge \ \pi_1(e) \in \beta[V_P] \ \wedge \ s_P(be(\pi_1(e))) \in V_{P^k} \\ 0 & \textit{otherwise} \end{cases}$
- $d_{P^k}(e) = \begin{cases} d_P(e) & \textit{if } e \in E_P \\ \pi_2(e) & \textit{if } e \in k \ \wedge \ \pi_2(e) \in S[V_P] \\ d_P(be(\pi_2(e)) & \textit{if } e \in k \ \wedge \ \pi_2(e) \in \beta[V_P] \ \wedge \ d_P(be(\pi_2(e))) \in V_{P^k} \\ 0 & \textit{otherwise} \end{cases}$

Remark 3.2.8. Since we still require that each boundary vertex occur at most once, there are only finitely many $*$-closures for any graph pattern P. One might consider allowing $*$-closure elements between two $*$-vertices, which would allow an infinite number of $*$-closures. We shall see in the next proposition that this is not necessary.

Proposition 3.2.9. *Let G be a graph. Let $(P \rightsquigarrow Q, \chi)$ be a graph pattern rewrite. For any instance (σ, B), closure k of σP, and matching $m : \sigma P \to G$, there exists a $*$-closure k' and $*$-matching $\widehat{m} : P^{k'} \to G$ such that*

$$G[m, (\sigma P)^k \to (\sigma_\chi Q)^k] = G[\widehat{m}, P^{k'} \rightsquigarrow Q^{k'}]$$

Proof. Assume without loss of generality that k contains a single closure (b_1, b_2) for $b_1, b_2 \in \beta[\sigma P]$. If b_1 and b_2 are in V_P, the result is immediate, letting $k' = k$. If $b_1 \in \beta[P]$ and $b_2 \in B$, let $s \in S[V_P]$ be the unique vertex such that $b_2 \in \sigma(s)$. Let $k' = \{(s, b_2)\}$. Let $\widehat{m}_e((s, b_2)) = m_e((b_1, b_2))$[1], and $\widehat{m}$ and m coincide on all other edges and vertices in $P^{k'}$. If both vertices are in B, let $k' = \{\}$. The loop is totally external to the $*$-rewrite, so it will be preserved as it is on the concrete rewrite. □

3.2.2 $*$-Critical Pairs

Definition 3.2.10. A **$*$-unification** U of two $*$-matchings $p : P \to U$ and $q : Q \to U$ is a graph pattern such that:

- $E_U = p_e(E_P) \cup q_e(E_Q)$

[1]Recall that these ordered pairs are the names of the edges created by performing the closure.

- $V_U = p_v(V_P) \cup q_v(V_Q)$
- $I[V_U] = p_v(I[V_P]) \cup q_v(I[V_Q])$
- $C[V_U] = p_v(C[V_P]) \cup q_v(C[V_Q])$

The forth condition ensures that the vertex types of the $*$-overlap are the "most general," i.e. all internal vertices are $*$-vertices, unless they are forced to be concrete by P or Q. ***-overlaps** are constructed for a pair of $*$-matchings the same way overlaps are constructed for matchings.

Definition 3.2.11. For graph pattern rewrites $P_1 \rightsquigarrow Q_1$ and $P_2 \rightsquigarrow Q_2$, a $*$-overlap $u : P_1 \to P_2$, a critical pair (C_1, C_2) is a pair of graph patterns where C_i is the result of $*$-rewriting $P_1 +_u P_2$ on $P_i \to Q_i$ $(i = 1, 2)$.

Lemma 3.2.12. *($*$-Critical Pair) If all $*$-critical pairs of a rewrite system are joinable, then a rewrite system is locally confluent.*

Proof. Follows immediately from 3.2.6. $\square$

Thus, to enumerate critical pairs of two graph pattern rewrite rules $p_1 : P_1 \rightsquigarrow Q_1$ and $p_2 : P_2 \rightsquigarrow Q_2$, one computes the $*$-overlaps for all $*$-closures k_i of P_i $(i = 1, 2)$.

3.3 Scalars

We shall represent scalars graphically as vertices connected to no edges. For the rewrite rules to follow, we shall only use two scalars, k and $1/k$. [2]

$$k := \triangle \quad \text{and} \quad 1/k := \triangledown$$

Scalars always occur on the RHS of rewrite rules, with the exception of the scalar rule, where $\emptyset$ is the empty graph:

$$\triangle\triangledown \to \emptyset$$

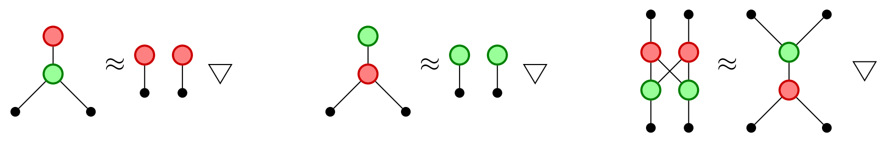
Just as scalar multiplication is associative, the scalar rewrite is always confluent with itself. We see this by looking at the only two critical pairs.

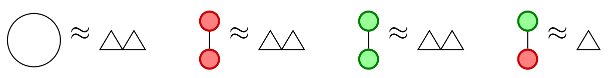
$$\triangle \leftarrow \triangle\triangledown\triangle = \triangle\triangledown\triangle \to \triangle \quad \text{and} \quad \triangledown \leftarrow \triangledown\triangle\triangledown = \triangledown\triangle\triangledown \to \triangledown$$

[2]For reasons described in e.g. [8], if A is a Hilbert space, then k is $\sqrt{D}$, where D is the dimension A.

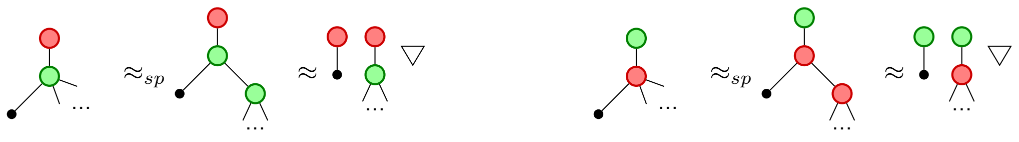
3.4 An Equational System

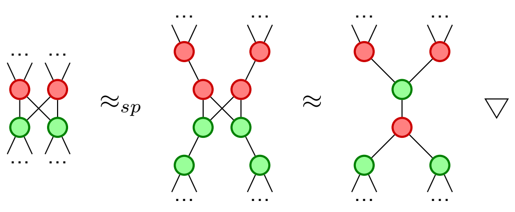
The equational system is $\approx_{sp}$ with the following additional rules:





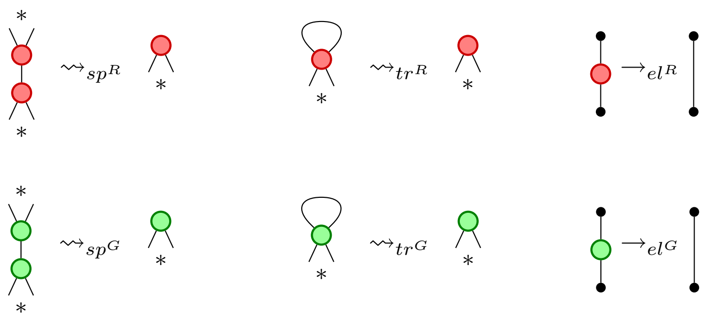
The top three rules can be generalised to apply to arbitrary spiders.

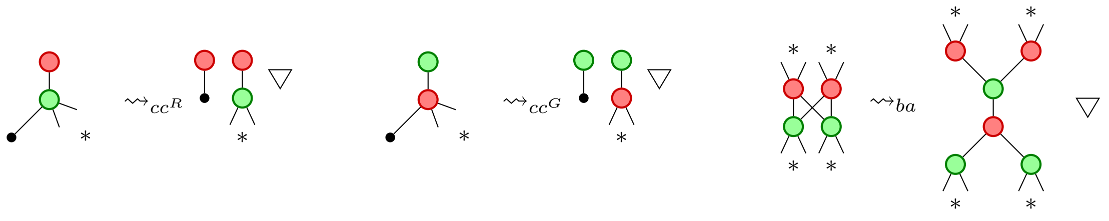


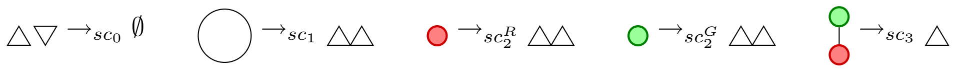


3.5 Rewrite Systems Q_2 and Q_3

Take the following to be Q_2, which is our naive first guess at a rewrite system:







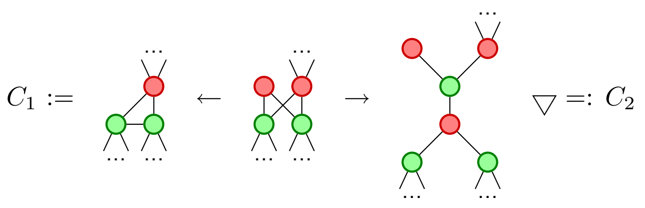
When it is clear from context, we will omit the superscripts ^{R} and ^{G} from the names of rules.

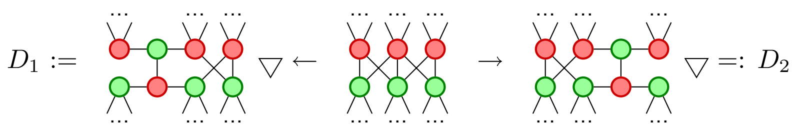
Definition 3.5.1. For a rewrite system R and a pair of types $r, g \in T$, R is said to be (r, g)-dual (or simply **colour-dual**) if for every rule $r \in R$ there is also a rule $r' \in R$, which is identical to r but with opposite colouring.

Proposition 3.5.2. *For a colour-dual rewrite system R, if G and H are graphs such that $G \to_R H$, then for their colour-dual graphs G' and H', it is also true that $G' \to H'$.*

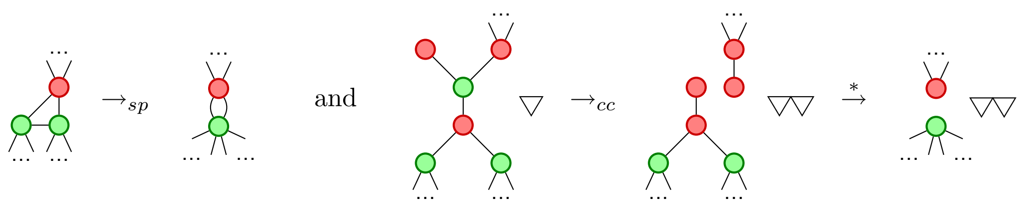
Remark 3.5.3. Q_2 is colour-dual.

As we begin to look at critical pairs, we soon find two counter-examples to confluence in this system. Let the critical pairs (C_1, C_2) and (D_1, D_2) be defined as:

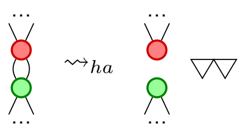




We begin by examining (C_1, C_2). Reducing to normal forms, we have:



Since these normal forms don't match, we join this critical pair by adding the completion:



This identity was noted in [8] along with the observation that complementary classical structures form a scaled Hopf algebra. Because of this, we shall call this rule ha.

The result of joining the critical pair (D_1, D_2) yields a non-terminating system. We will explore this system more in a later section. For now, we consider the (less powerful) rewrite system Q_3 obtained from replacing ba in Q_2 with ha.

3.5.1 Properties of the System without Bi-Algebra (Q_3)

Remark 3.5.4. Since Q_2 is colour-dual and ha is the colour-dual of itself, Q_3 is colour-dual.

Every rule in Q_3 decreases the complexity of a graph, so it stands to reason that Q_3 should be terminating. To prove this, we shall enunciate what we mean by complexity.

Definition 3.5.5. For a graph G, the **edge complexity** $ec(G)$ is

$$\Sigma\{|\,in(v) \cup out(v)| \,:\, v \in V_G, |\,in(v) \cup out(v)| > 1\}$$

We define $\succ$ is the lexicographic order of edge cardinality and edge complexity. I.e.:

$$G \succ H \Leftrightarrow (|E_G| > |E_H| \vee (|E_G| = |E_H| \wedge ec(G) > ec(H)))$$

Proposition 3.5.6. *$\succ$ is a reduction order on Q_3.*

Proof. $\succ$ is a lexicographic order of two integer values, so it is a strict order. Since $E_G \geq 0$ and $ec(G) \geq 0$ for all G, it is well-founded. Now, consider G and H such that $G \to H$. If G is rewritten using any rule besides cc, the number of edges decreases, so assume $G \to_{cc} H$. Assume that the $*$-vertex in cc matches n nodes. For any n, $|E_G| = |E_H|$, so we need to look at the edge complexity. If n is 0, then $ec(H) = ec(G) - 2$. If $n > 0$, then $ec(H) = ec(G) - 1$. So, $G \succ H$. $\square$

Remark 3.5.7. Since Q_3 has a reduction order, it is terminating. By Newman's lemma, we need only local confluence to show that Q_3 is confluent.

Definition 3.5.8. We say a $*$-critical pair (C_1, C_2) is **trivial** if $C_1 = D_1^k$ and $C_2 = D_2^k$ for some joinable critical pair (D_1, D_2), or if it arose from a $*$-overlap u of $P_1 \rightsquigarrow Q_1$ and $P_2 \rightsquigarrow Q_2$ where

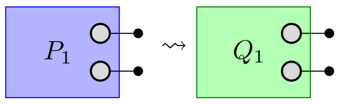
(i) u is empty or total

(ii) arose from a $*$-unification of $P_i \rightsquigarrow Q_i$ $(i = 1, 2)$, where no two boundary vertices are connected that shares only boundary edges

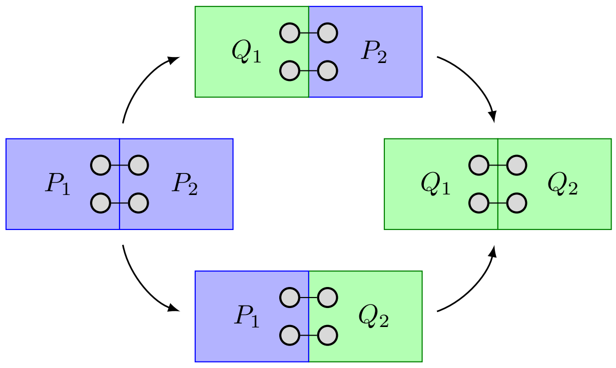
(iii) it shares at most one vertex of each colour

Proposition 3.5.9. *All trivial overlaps are joinable.*

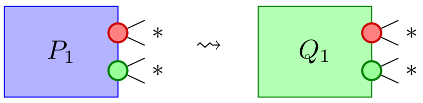
Proof. The closure case follows immediately from lemma 1.4.21. For case (i), if u is empty, there is no overlap. If u is total, then $C_1 = C_2$. Case (ii) and (iii) are easiest to see visually. Let $P_1 \rightsquigarrow Q_1$ be a pattern rewrite rule, with some boundary vertices:



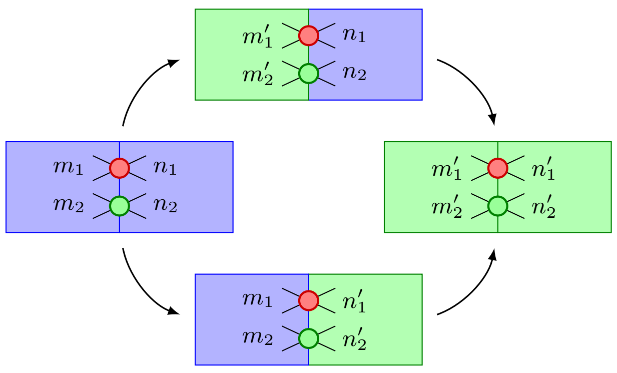
Where "○" means any vertex, and the vertices are not necessarily distinct. Note that the internal vertices may change, but the boundary edges are always preserved. Any two such rules sharing only boundary vertices is joinable:



Case (iii) follows similarly. Let $(P_1 \rightsquigarrow Q_1, \chi_1)$ be a pattern rewrite rule, with (up to) one $*$-vertex of each colour:



Consider a pair of pattern rewrite rules $(P_i \rightsquigarrow Q_i, \chi_i)$ for $i = 1, 2$. For some graph G, let $m : P_1 \to G$ and $n : P_2 \to G$ be $*$-matchings. For a $*$-vertex $v \in S[P_1]$, we can distinguish in G the internal edges $IE_v = m_e(in_{P_1}(v))$ from the external edges $EE_v = in_G(m_e(v)) \backslash m_e(in_{P_1}(v))$. The edges in IE_v are in the image of P_1, so they can be modified by the rewrite, but the edges in EE_v are (mostly) preserved. For $e \in EE_v$, the source of e is unchanged and the destination of e is changed to $\chi_1(v)$. Since χ_1 respects types, if e was connected to a red (resp. green) vertex before the rewrite, it will be connected to a red (resp. green) vertex after the rewrite. Since χ_1 need not be injective, the $*$-rewrite may merge some $*$-vertices, but if we restrict the overlap of m and n to at most one $*$-vertex of each colour, this is irrelevant. In the following diagram, m_1, m_2 label edges that are internal to P_1 and n_1, n_2 label edges that are internal to P_2.



□

Proposition 3.5.10. *The rules of Q_3 generate the following numbers of non-trivial critical pairs:*

	sp^R	sp^G	tr^R	tr^G	cc^R	cc^G	el^R	el^G	ha
sp^R	0	0	0	0	0	1	1	0	0
sp^G		0	0	0	1	0	0	1	0
tr^R			0	0	0	1	1	0	0
tr^G				0	1	0	0	1	0
cc^R					2	0	0	1	1
cc^G						2	1	0	1
el^R							0	0	1
el^G								0	1
ha									1

Proof. (sp^R) This rule consists of an edge and two $*$-vertices. As a result of proposition 3.5.9, we shall only consider overlaps that share at least one edge or both vertices. Every non-trivial overlap sp^R has with itself is joinable by the vertical symmetry of the rule. Since no other rewrite has two red vertices, we shall only consider those that share an edge. We rule out any other rewrite rule that contains only green vertices. sp^R has only trivial overlaps with tr^R and no overlaps with cc^R. There is one edge in cc^G that admits a valid overlap, and one edge in el^R. The only overlap with ha is trivial.

(tr^R) A non-trivial overlap with tr^R must share the loop edge. Since tr^R is the only rule with a self-loop, we must look at $*$-closures of other rules to obtain possible overlaps. Again we can rule out rewrites that have only green vertices. The only overlap with itself is trivial. The only available closures on cc^R are on a green vertex. cc^G has one closure that yields an overlap, and so too does el^R. The only possible overlaps with ha are trivial.

(cc^R) Consider overlaps of cc^R with itself. If it shares just the $*$-vertex, the overlap is trivial. If it shares the vertex with incidence 1, it must also share the associated edge. Thus, we only need to look at cases where cc^R shares at least one edge. cc^R cannot overlap with cc^G, because the incidences of their respective red and green vertices are incompatible. Since the incidence of the only el^R vertex is 2, it has no overlap with cc^R. el^G has exactly one overlap, on to the (green) $*$-vertex of cc^R. cc^R could not share its concrete vertex (or upper edge, by homomorphism) with ha. There is one unique way (up to symmetry) that it could share its lower edge with ha.

(el^R) This rewrite can only share boundary edges with itself, so all overlaps are trivial. Since its one vertex has incidence 2, it has one unique overlap with ha.

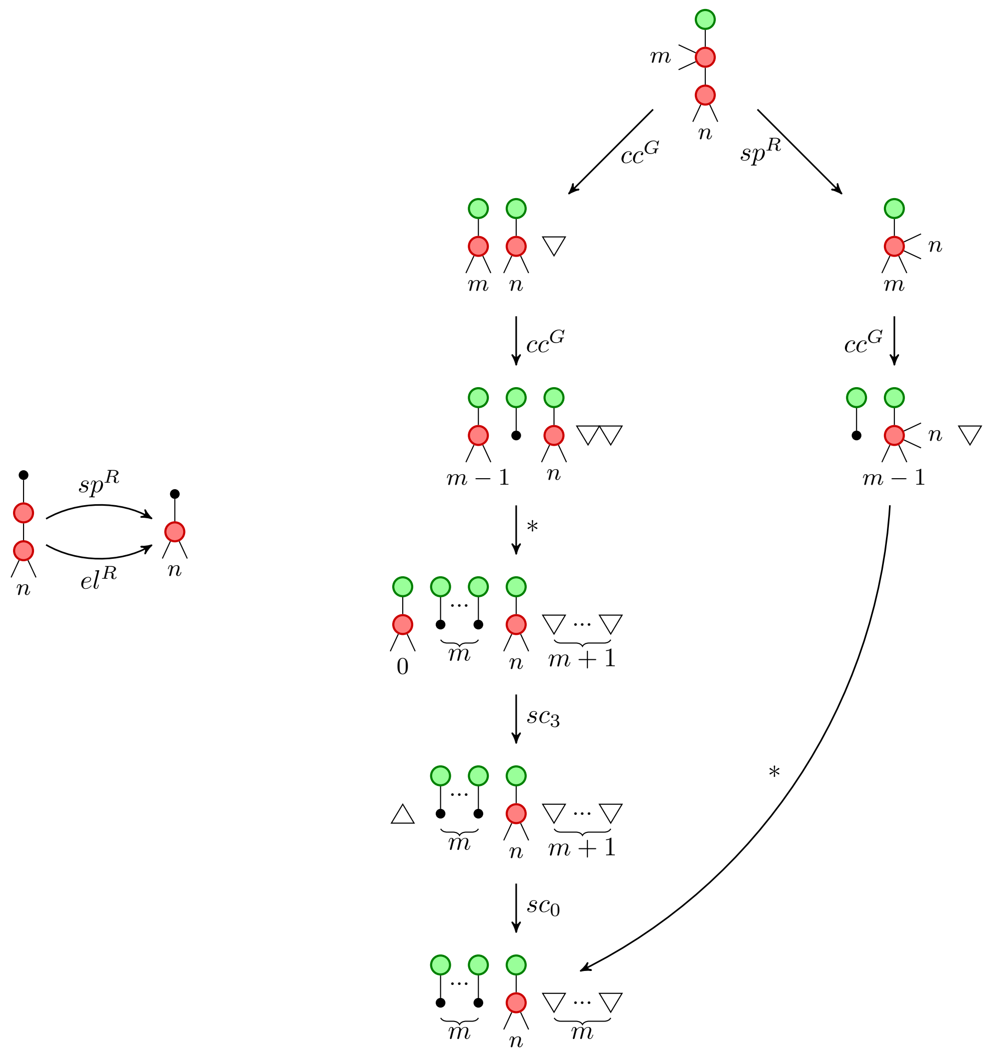
(ha) To be non-trivial, an overlap of ha with itself must share an edge. There is one way this can happen. If it shares two edges, it is a total overlap (by homomorphism), which is trivial.

(sp^G), (tr^G), (cc^G), and (el^G) all follow from above, swapping "red" and "green". □

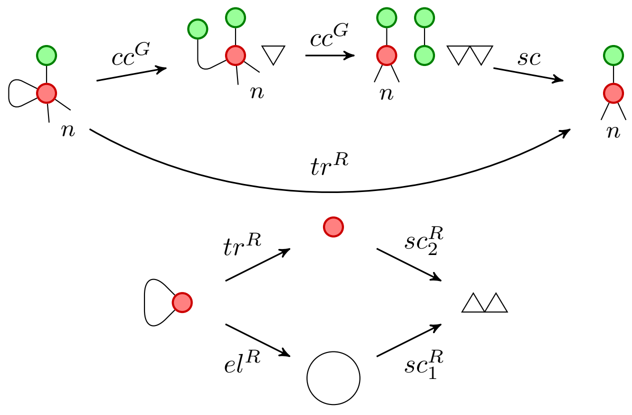
Proposition 3.5.11. *Q_3 is locally confluent.*

Proof. We do this by showing joinability of the non-trivial critical pairs given above.

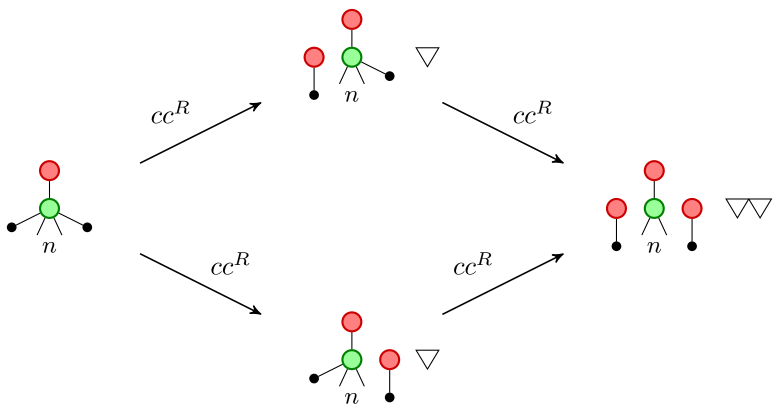
Critical pairs with sp^R:

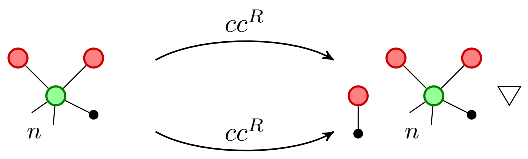


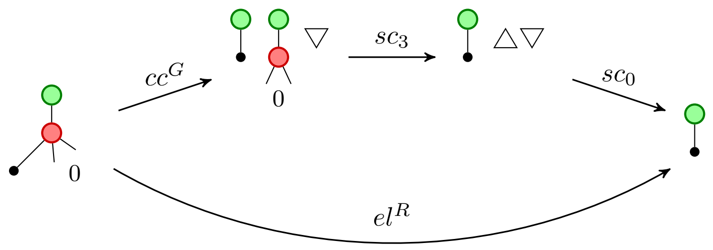
Critical pairs with tr^R:

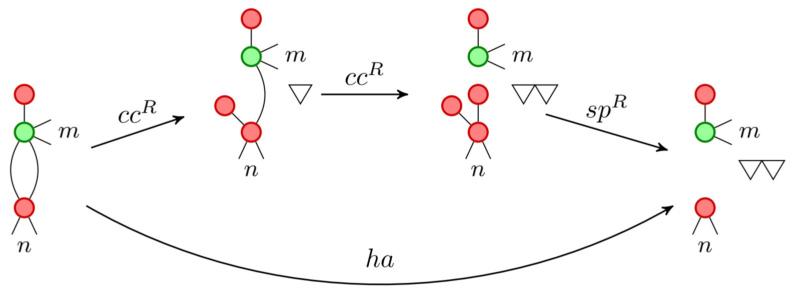


Critical pairs with cc^R:

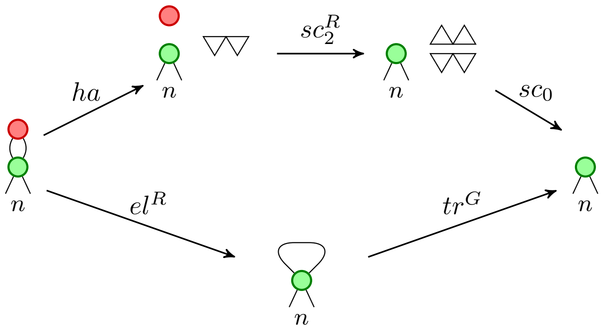




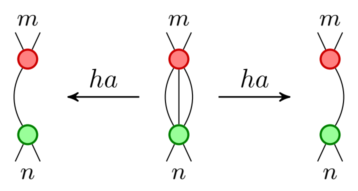




Critical pairs with el^R:



Critical pairs with ha:



All of the remaining critical pairs are joinable by colour-duality. By the critical pair lemma, Q_3 is locally confluent. □

Q_3 is locally confluent and terminating, so it is confluent by Newman's lemma. It is confluent and terminating, so it is convergent.

Proposition 3.5.12. *The normal forms $\mathcal{N} \subset \mathcal{G}$ with respect to Q_3 are graphs that*

(i) contain no spiders of incidence 0

(ii) contain no empty edges

(iii) can contain k or $1/k$, but not both

(iv) have the property that all spiders of incidence 1 are connected only to boundary vertices

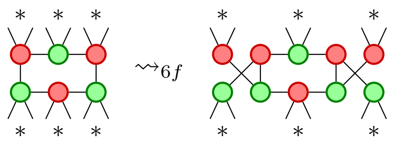
(v) are bipartite on colour

(vi) only contains even cycles of size four or more

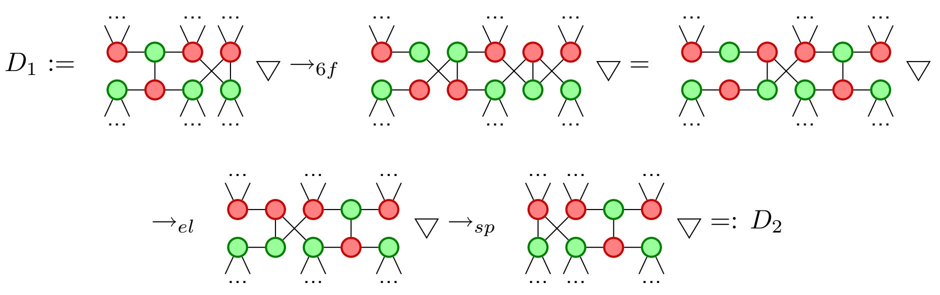
Proof. No rewrite in Q_3 applies to graphs with the given properties. For the converse, assume one of the properties does not hold. If one of the first three conditions is not true, the appropriate scalar rewrite applies. Let v be a red spider (or dually, a green spider) with incidence 1. If it connected to another red spider, sp^R applies. If it is connected to a green spider of incidence 1, sc_3 applies. If it is connected to a green spider of incidence 2, cc^R applies. If the graph is no bipartite on colour, either a spider is connected to itself, or to another spider of the same colour. In these cases, tr and sp apply, respectively. If there exists a 1-cycle (a loop), tr applies. If there exists a 2-cycle, ha applies. If there exists any other odd cycle, the graph must not be bipartite on colour, so sp applies. □

3.6 Bi-algebra Completion and 6-Cycles

In the previous section, we showed Q_2 has a non-joinable critical pair (D_1, D_2) induced by an overlap from ba to ba. We could admit $D_1 \to D_2$ as a rewrite, but instead we chose a slightly simpler rule:



We refer to this as the “six-flip” rule, because it flips four of the edge incidences in a six-cycle. It closes the critical pair:



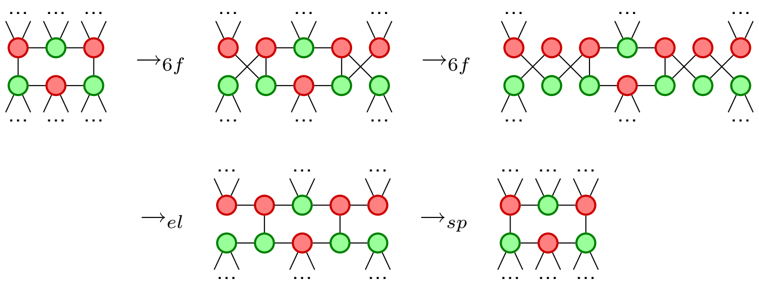
Proposition 3.6.1. $\rightarrow_{6f} \subseteq \overset{*}{\leftrightarrow}_{Q_2}$.

Proof. Similar to the derivation of (D_1, D_2). □

Let Q_4 be the rewrite system Q_3, with the additional rules ba and $6f$.

Proposition 3.6.2. *Q_4 is non-terminating.*

Proof. The rule $6f$ is nearly its own inverse.



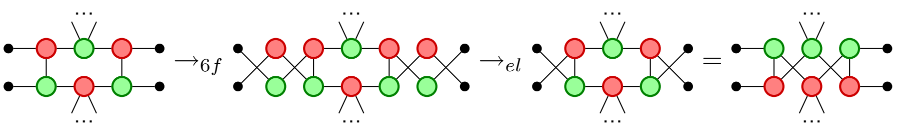
□

Proposition 3.6.3. *Q_4 is locally confluent.*

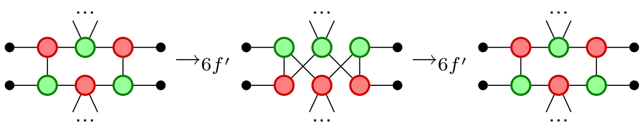
Proof. Since all the rules in Q_3 join with each other, and $6f$ can be used to derive its own inverse, it suffices to show ba joins with the other rules. This can be done with critical pair enumeration, as in proposition 3.5.11. □

Since Q_4 is not terminating, we cannot apply Newman's lemma. It remains an open question whether Q_4 is confluent.

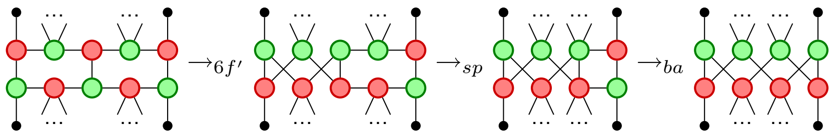
To see some of the properties of Q_4, we derive a more specific kind of six-flip rule:



Let this new rule be $6f'$, and note:



ha removes 2-cycles and *ba* disconnects 4-cycles. While $6f$ doesn't directly disconnect 6-cycles, it does allow bordering 6-cycles to merge.



More specifically, the 6-cycle on the left "donates" an edge to the 6-cycle on the right, making it a 4-cycle. It can donate an edge to a bordering cycle of any size, so if multiple 6-cycles are adjacent to a larger cycle, they can all potentially donate edges until a 2- or 4-cycle is created. This leads to a possible terminating rewrite strategy.

(i) Perform as many reductions in $Q_4 - \{6f\}$ as possible.

(ii) Identify clusters of connected cycles containing at least two 6-cycles.

(iii) Attempt to form 2- and 4-cycles by applying $6f$.

(iv) Repeat until no more reductions in $Q_4 - \{6f\}$ can be found.

If this strategy does indeed yield normal forms modulo $6f$, they will look much like the normal forms of Q_3, with the additional constraints that all cycles must be of size 6 or more, and 6-cycles must be suitably isolated. It is a topic of ongoing research to find sufficient isolation conditions such that 6-cycles cannot yield additional rewrites via $6f$.

Chapter 4

Conclusion and Future Work

4.1 Summary of Results

We first introduced the notion of graph matching and rewriting with types, as well as a version of the critical pair lemma that allows us to seek local confluence in graph rewrite systems. We then applied this method to construct a rewrite system Q_1 that had two types, red vertices and boundary vertices. It contains rewrite rules that reflect the monoidal, co-monoidal, isometric, and frobenius identities on a single classical structure. These rules were used to give a proof of the spider theorem for Q_1, which states that any connected graph of a single classical structure is uniquely determined by the number of inputs and outputs.

For the next rewrite system, Q_2, we made two generalisations. The first was to allow for undirected graphs, and the second was to add an additional type for green vertices. Q_2 contains rules implied by the spider theorem for both colours, as well as rules derived from the interaction of complementary classical structures.

After exploring some of the critical pairs of Q_2, we noted that this system is not confluent. We introduced a completion ha into the system and removed the bialgebra rewrite to get the system Q_3. We showed that this system is terminating and confluent, but it is clearly not as powerful as Q_2.

The final rewrite system we explored was Q_4, which re-introduced ba into Q_3, as well as a completion $6f$. We showed that this system is locally confluent but not terminating, so it may not be confluent.

4.2 Future Work

We are currently working to mechanise rewriting on two-colour graphs. We have a (partially-implemented) tool that performs rewrites manually[1], and we are pursuing methods of automatically performing rewrites and seeking out normal forms. An automated tool could also provide a framework for performing mechanised critical pair analysis and completion. With such a tool, we may be able to input a rewrite system like Q_2 and compute a confluent extension using a Knuth-Bendix completion algorithm [16].

On the theoretical side, the clear next step is to prove confluence for Q_4 or a similar system. Next, it would be useful to demonstrate a rewrite strategy for Q_4 that (a) yields normal forms, modulo $6f$ and (b) can do so with a feasible time complexity. There are also additional graph primitives that would be useful to include in the rewrite system, namely Hadamard gates and phase angles. By augmenting two-colour graphs with angle vertices, one can represent all of the unbiased points in either classical structure, which greatly increases the power of the graphical language. Hadamard gates provide a change of basis and introduce a natural duality relationship between the two colours, which also increases the power of the language. Coecke and Duncan describe these additions in-depth in [8], and go on to show that a pair of complementary classical structures and the Hadamard gate suffice to construct the "controlled-X" and "controlled-Z" gates, which are computationally universal. Since quantum gates can therefore be expressed in terms of classical structures, this and related works might prove a useful new way to study circuit models of quantum mechanics.

[1] http://dream.inf.ed.ac.uk/projects/quantomatic

Bibliography

[1] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. *Proceedings from LiCS*, quant-ph, Feb 2004.

[2] Andrea Asperti and Giuseppe Longo. Categories, types, and structures. *wiki.ittc.ku.edu*, Jan 1991.

[3] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* 1998.

[4] Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. Diagram techniques for confluence. *Information and Computation*, 141(2):172–204, 1998.

[5] Alonzo Church and J Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472—482, 1936.

[6] Bob Coecke. De-linearizing linearity: Projective quantum axiomatics from strong compact closure. *arXiv*, quant-ph, Jun 2005.

[7] Bob Coecke. Kindergarten quantum mechanics. *arXiv*, quant-ph, Oct 2005.

[8] Bob Coecke and Ross Duncan. Interacting quantum observables. *ICALP*, pages 298—310, Mar 2008.

[9] Bob Coecke and Dusko Pavlovic. Quantum measurements without sums. *arXiv*, quant-ph, Aug 2006.

[10] A Corradini, U Montanari, F Rossi, H Ehrig, R Heckel, and M Lowe. Algebraic approaches to graph transformation, part i: Basic concepts and double pushout approach. *Handbook of Graph Grammars and Computing by Graph Transformation*, 1:163–245, 1997.

[11] N Lafaye de Micheaux and C Rambaud. Confluence for graph transformations. *Theoretical Computer Science*, 154(2):329–348, 1996.

[12] Nachum Dershowitz. Open. closed. open. *LNCS*, 3647:376–393, 2005.

[13] Lucas Dixon and Ross Duncan. Extending graphical representations for compact closed categories with applications to symbolic quantum computation. *AISC/MKM/Calculemu*, pages 77—92, Jun 2008.

[14] Ross Duncan. Types for quantum computing. page 175, Jun 2007.

[15] Bernhard Gramlich. Confluence without termination via parallel critical pairs. *Colloquium on Trees in Algebra and Programming*, pages 211–225, 1996.

[16] Gérard Huet. A complete proof of correctness of the knuth-bendix completion algorithm. *Rapports de Recherche*, 25:15, 1980.

[17] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM (JACM)*, 27(4):797–821, 1980.

[18] Max Kelly and Miguel L Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.

[19] Peter Selinger. Dagger compact closed categories and completely positive maps (extended abstract). *Electronic Notes in Theoretical Computer Science*, 170:139–163, 2007.

[20] Pedro Pablo Perez Velasco and Juan de Lara. Matrix approach to graph transformation: Matching and sequences. *LNCS*, 4178:122, 2006.