# A FINITE AXIOMATISATION OF FINITE-STATE AUTOMATA USING STRING DIAGRAMS

ROBIN PIEDELEU AND FABIO ZANASI

University College London, United Kingdom
*e-mail address*: r.piedeleu@ucl.ac.uk

University College London, United Kingdom
*e-mail address*: f.zanasi@ucl.ac.uk

ABSTRACT. We develop a fully diagrammatic approach to finite-state automata, based on reinterpreting their usual state-transition graphical representation as a two-dimensional syntax of string diagrams. In this setting, we are able to provide a complete equational theory for language equivalence, with two notable features. First, the proposed axiomatisation is finite. Second, the Kleene star is a derived concept, as it can be decomposed into more primitive algebraic blocks.

## 1. INTRODUCTION

Finite-state automata are one of the most studied structures in theoretical computer science, with an illustrious history and roots reaching far beyond, in the work of biologists, psychologists, engineers and mathematicians. Kleene [28] introduced regular expressions to give finite-state automata an algebraic presentation, motivated by the study of (biological) neural networks [36]. They are the terms freely generated by the following grammar:

$$e, f ::= e + f \mid ef \mid e^* \mid 0 \mid 1 \mid a \in A \tag{1.1}$$

Equational properties of regular expressions were studied by Conway [16] who introduced the term *Kleene algebra*: this is an idempotent semiring with an operation $(-)^*$ for iteration, called the (Kleene) star. The equational theory of Kleene algebra is now well-understood, and multiple complete axiomatisations, both for language and relational models, have been given. Crucially, Kleene algebra is not finitely-based: no finite equational theory can appropriately capture the behaviour of the star [41]. Instead, there are purely equational infinitary axiomatisations [32, 5] and finitary implicational theories, like that of Kozen [29].

Since then, much research has been devoted to extending Kleene algebra with operations capturing richer patterns of behaviour, useful in program verification. Examples include conditional branching (Kleene algebra with tests [30], and its recent guarded version [43]), concurrent computation (CKA [22, 26]), and specification of message-passing behaviour in networks (NetKAT [1]).

---

*Key words and phrases:* string diagrams, finite-state automata, symmetric monoidal category, complete axiomatisation.

The meta-theory of the formalisms above relies on a three step methodology: (1) given an operational model (e.g., finite-state automata), (2) devise a syntax (regular expressions) that is sufficiently expressive to capture the class of behaviours of the operational model (regular languages), and (3) find a complete axiomatisation (Kleene algebra) for the given semantics.

In this paper, we open up a direct path from (1) to (3). Instead of thinking of automata as a combinatorial model, we formalise them as a bona-fide (two-dimensional) syntax, using the well-established mathematical theory of *string diagrams* for monoidal categories [42]. This approach lets us axiomatise the behaviour of automata directly, freeing us from the necessity of compressing them down to a one-dimensional notation like regular expressions.

This perspective not only sheds new light on a venerable topic, but has significant consequences. First, as our most important contribution, we are able to provide a *finite and purely equational* axiomatisation of finite-state automata, up to language equivalence. This does not contradict the impossibility of finding a finite basis for Kleene algebra, as the algebraic setting is different: our result gives a finite presentation as a symmetric monoidal category, while the impossibility result prevents any such presentation to exist as an algebraic theory (in the standard sense). In other words, there is no finite axiomatisation based on terms (*tree*-like structures), but we demonstrate that there is one based on string diagrams (*graph*-like structures).

Secondly, embracing the two-dimensional nature of automata guarantees a strong form of compositionality that the one-dimensional syntax of regular expressions does not have. In the string diagrammatic setting, automata may have multiple inputs and outputs and, as a result, can be decomposed into subcomponents that retain a meaningful interpretation. For example, if we split the automata below left, the resulting components are still valid string diagrams within our syntax, below right:



$$\tag{1.2}$$
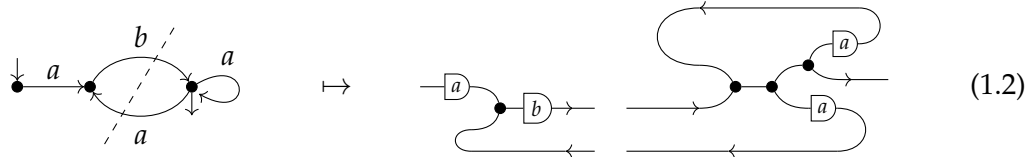
In line with the compositional approach, it is significant that the Kleene star can be decomposed into more elementary building blocks (which come together to form a feedback loop):

$$e^* \quad \mapsto$$



$$\tag{1.3}$$
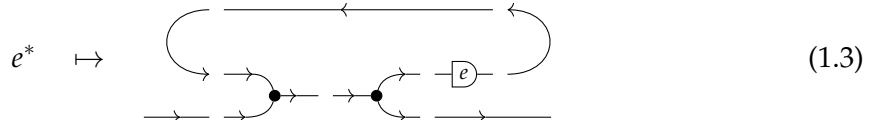
This opens up for interesting possibilities when studying extensions of Kleene algebra within the same approach— we elaborate on this in Section 6.

Finally, we believe our proof of completeness is of independent interest, as it relies on fully diagrammatic reformulation of Brzozowski's minimisation algorithm [13]. In the string diagrammatic setting, the symmetries of the equational theory give this procedure a particularly elegant and simple form. Because all of the axioms involved in the determinisation procedure come with a dual, a co-determinisation procedure can be defined immediately by simply reversing the former. This reduces the proof of completeness to a proof that determinisation can be performed diagrammatically. Moreover, note that our completeness proof goes through a richer language, with additional algebraic operations

that are adjoint (in a sense that we explain in Section 3) to those that allow us to express standard automata. Within this extended language, we are able to derive a completeness result for (diagrams corresponding to) automata, but leave open the completeness of the full language. We will come back to this point in Section 6.

This is not the first time that automata and regular languages are recast into a categorical mould. The *iteration theories* [6] of Bloom and Ésik, *sharing graphs* [20] of Hasegawa or *network algebras* [45] of Stefanescu are all categorical frameworks designed to reason about iteration or recursion, that have found fruitful applications in this domain. They are based on a notion of parameterised fixed-point which defines a categorical *trace* in the sense of [25]. While our proposal bears resemblance to (and is inspired by) this prior work, it goes beyond in one fundamental aspect: it is the first to give a *finite* complete axiomatisation of automata up to language equivalence.

A second difference is methodological: our syntax does not feature any primitive for iteration or recursion. In particular, the star is a derived concept, in the sense that it is decomposable into more elementary operations (1.3). Categorically, our starting point is a compact-closed rather than traced category.

We elaborate further on the relation between ours and existing work in Section 6.

**Conference version.** This work is based on the conference paper [38]. It amends a mistaken completeness claim made in that paper and proposes a new approach to the same question, based on a different syntax. We detail the relationship between the two papers in Section 6 and where clarifications are necessary in the main text.

## 2. SYNTAX AND SEMANTICS

Following a standard methodology (recalled in Appendix A), we will define two symmetric monoidal categories (SMCs), one serving as syntax, the other as semantics. Moreover, to guarantee a compositional interpretation, we will define a symmetric monoidal functor between the two.

**Syntax.** We fix an alphabet $\Sigma$. We call Syn the strict SMC freely generated by the following objects and morphisms:

- two generating objects ▶ ('right') and ◀ ('left') with their identity morphisms depicted respectively as ——→—— and ——←——.
- the following generating morphisms, depicted as *string diagrams* [42]:

$$\rightarrow\!\!\varprec_{\rightarrow}^{\rightarrow} \quad \rightarrow\!\bullet \quad \overset{\rightarrow}{\supset}\!\bullet\!\succ\!\!- \quad \bullet\!\succ\!\!- \quad \overset{\rightarrow}{\supset} \quad \subsetneq \quad -\boxed{a}\!- \quad (a \in \Sigma) \tag{2.1}$$

$$\overset{\rightarrow}{\supset}\!\succ\!\!- \quad \circ\!\succ\!\!- \quad \rightarrow\!\!\varsubsetneq_{\rightarrow}^{\rightarrow} \quad \rightarrow\!\circ \tag{2.2}$$

**Semantics.** We first define the semantics for string diagrams simply as a mapping from the set of generators to relations between tuples of languages over $\Sigma$, and then discuss how to extend it to a functor from Syn to a category that we will define below. Let $\mathcal{L}_\Sigma := 2^{\Sigma^\star}$. A diagram with $m$ ports on the left and $n$ on the right, is interpreted as a subset of $\mathcal{L}_\Sigma^m \times \mathcal{L}_\Sigma^n$.

$$\left[\!\left[ \to\!\overset{\to}{\bullet\,\raisebox{1pt}{$\subset$}_{\to}} \right]\!\right] = \left\{ \left( L, (K_1, K_2) \right) \mid L \subseteq K_i,\ i = 1, 2 \text{ and } L, K_1, K_2 \in \mathcal{L}_\Sigma \right\}$$

$$\left[\!\left[ \overset{\to}{\raisebox{1pt}{$\supset$}_{\to}}\!\bullet\!\rightarrowtail \right]\!\right] = \left\{ \left( (L_1, L_2), K \right) \mid L_i \subseteq K,\ i = 1, 2 \text{ and } L_1, L_2, K \in \mathcal{L}_\Sigma \right\}$$

$$\left[\!\left[ \to\!\bullet \right]\!\right] = \{ (L, \bullet) \mid L \in \mathcal{L}_\Sigma \} \qquad \left[\!\left[ \subset_{\raisebox{-2pt}{}} \right]\!\right] = \{ (\bullet, (L, K)) \mid L \subseteq K \mid L, K \in \mathcal{L}_\Sigma \}$$

$$\left[\!\left[ \bullet\!\rightarrowtail \right]\!\right] = \{ (\bullet, K) \mid K \in \mathcal{L}_\Sigma \} \qquad \left[\!\left[ \overset{\raisebox{-2pt}{$\supset$}}{} \right]\!\right] = \{ ((L, K), \bullet) \mid K \subseteq L,\ L, K \in \mathcal{L}_\Sigma \}$$

$$\left[\!\left[ -\!\boxed{a}\!- \right]\!\right] = \{ (L, K) \mid La \subseteq K,\ L, K \in \mathcal{L}_\Sigma \} \qquad (a \in \Sigma)$$

$$\left[\!\left[ -\!\!\longrightarrow\!\!- \right]\!\right] = \{ (L, K) \mid L \subseteq K,\ L, K \in \mathcal{L}_\Sigma \}$$

$$\left[\!\left[ -\!\!\longleftarrow\!\!- \right]\!\right] = \{ (L, K) \mid K \subseteq L,\ L, K \in \mathcal{L}_\Sigma \}$$

$$\left[\!\left[ \overset{\to}{\raisebox{1pt}{$\supset$}_{\to}}\!\rightarrowtail \right]\!\right] = \{ ((L_1, L_2), K) \mid L_1 \cap L_2 \subseteq K,\ L_1, L_2, K \in \mathcal{L}_\Sigma \} \qquad \left[\!\left[ \circ\!\rightarrowtail \right]\!\right] = \{ (\bullet, \Sigma^\star) \} \qquad (2.3)$$

$$\left[\!\left[ \to\!\overset{\to}{\raisebox{1pt}{$\subset$}_{\to}} \right]\!\right] = \{ (L, (K_1, K_2)) \mid L \subseteq K_1 \cup K_2,\ L, K_1, K_2 \in \mathcal{L}_\Sigma \} \qquad \left[\!\left[ \to\!\circ \right]\!\right] = \{ (\varnothing, \bullet) \} \qquad (2.4)$$

In a nutshell, the generating diagrams denote operations that relate tuples of languages: $\to\!\overset{\to}{\bullet\subset_\to}$ represents copying, $\to\!\bullet$ discarding, $\subset$ and $\supset$ feed back outputs into inputs and vice-versa, and $-\boxed{a}-$ represents the action of each letter of $\Sigma$ on the set of languages, by concatenation on the right. These are the generators that allow us to encode automata, as we will see in Section 4. The other generators, $\overset{\to}{\supset_\to}\!\rightarrowtail, \circ\!\rightarrowtail, \to\!\overset{\to}{\subset_\to}, \to\!\circ$, represent operations that are adjoint to their black counterparts, in a sense that we will explain in Section 3. The directed syntax highlights the dual roles played by the two generating objects, representing inclusion and reverse inclusion of languages respectively.

**Remark 2.1.** A word of warning: in the conference paper [38] on which this work is based, we use a different three sorted syntax, with an additional red wire denoting the set of regular expressions. These acted on the set of languages via a white node whose appearance is reminiscent of the white nodes of the syntax above. However, the reader should not confuse them—their semantics are totally unrelated and, as we will see, so are their equational properties.

In order for the mapping $\llbracket \cdot \rrbracket$ to be functorial, we now introduce a suitable target SMC for the semantics. Interestingly, this will not be the category Rel of sets and relations: indeed, the identity morphisms $\longrightarrow$ and $\longleftarrow$ are not interpreted as identities of Rel (since they denote the order on the set of languages). Instead, the semantic domain will be the category $\mathsf{Prof}_\mathbb{B}$ of *Boolean(-enriched) profunctors* [17] (also variously called relational profunctors [23] or weakening relations [37] in the literature).

**Definition 2.2.** Given two preorders $(X, \leq_X)$ and $(Y, \leq_Y)$, a *Boolean profunctor* $R : X \to Y$ is a relation $R \subseteq X \times Y$ such that if $(x, y) \in R$ and $x' \leq_X x$, $y \leq_Y y'$ then $(x', y') \in R$.

Preorders and Boolean profunctors form a SMC $\mathsf{Prof}_\mathbb{B}$ with composition given by relational composition. The identity for an object $(X, \leq_X)$ is the order relation $\leq_X$ itself. The monoidal product is the usual product of preorders, where $(x, y) \leq (x', y')$ iff $x \leq_X x'$ and

$y \leq_Y y'$. For more details on Boolean profunctors, including applications to engineering design, we refer the reader to [17, Chapter 4]. Since relations can be ordered by inclusion in a way that is compatible with composition, they form a *bicategory*, and so does $\mathsf{Prof}_{\mathbb{B}}$. The bicategorical structure is rather simple as, for any two morphisms, the set of 2-cells between them forms a partial order. This also means that $\mathsf{Prof}_{\mathbb{B}}$ is an *order-enriched* (1-)category. In fact, it is a *Cartesian bicategory* [14].

**Remark 2.3.** Note that every monotone map $f : (X, \leq_X) \to (Y, \leq_Y)$ can be turned into a monotone relations in two different ways: the relation $Yf := \{(x,y) \mid f(x) \leq y\}$ with type $Yf : (X, \leq_X) \to (Y, \leq_Y)$ and $Y^{op}f := \{(y,x) \mid y \leq f(x)\}$ with type $(Y, \leq_Y) \to (X, \leq_X)$. This implies that $\mathsf{Prof}_{\mathbb{B}}$ contains (two different copies of) the SMC of monotone maps as a monoidal sub-category.

The features of our diagrammatic syntax reflect the rich structure of the profunctor semantics. Indeed, the order relation is built into the wires $\longrightarrow$ and $\longleftarrow$. The two possible directions represent the identities on the set of languages ordered by inclusion, and on the same set equipped with the reverse order, respectively.

**Proposition 2.4.** $[\![\cdot]\!]$ *defines a symmetric monoidal functor of type* $\mathsf{Syn} \to \mathsf{Prof}_{\mathbb{B}}$.

*Proof.* It suffices to check that the interpretation of all generators define Boolean profunctors. It is clear that all generators satisfy the condition of Definition 2.2. For example, the action generator $-\boxed{a}-$ is a Boolean profunctor: if $(L, K)$ are such that $La \subseteq K$ and, moreover we have $L' \subseteq L$ and $K \subseteq K'$, then $L'a \subseteq La \subseteq K \subseteq K'$ by monotony of concatenation of languages. $\qquad\square$

In particular, because $\mathsf{Syn}$ is free, we can unambiguously assign meaning to any composite diagram from the semantics of its components using composition and the monoidal product in $\mathsf{Prof}_{\mathbb{B}}$:

$$[\![c\,;d]\!] = \left[\!\!\left[\; ^X\!\boxed{c}^Y\!\boxed{d}^Z \;\right]\!\!\right] = \{(L,K) \mid \exists M\,(L,M) \in [\![-\boxed{c}-]\!], (M,K) \in [\![-\boxed{d}-]\!]\}$$

$$[\![c_1 \oplus c_2]\!] = \left[\!\!\left[\; \begin{matrix} ^{X_1}\!\boxed{c_1}^{Y_1} \\ ^{X_2}\!\boxed{c_2}^{Y_2} \end{matrix} \;\right]\!\!\right] = \{((L_1,L_2),(K_1,K_2)) \mid (L_i,K_i) \in [\![-\boxed{c_i}-]\!], i = 1,2\}$$

Single wires labelled by a list $X$ of generating objects (here, $\blacktriangleright$ and $\blacktriangleleft$) represent $|X|$ parallel ordered wires, labelled from top to bottom with the elements of $X$.

**Example 2.5.** We include here a worked out example to show how to compute the behaviour of a composite diagram which, as we will see, represents (the action by concatenation of) the regular language $a^* = \{\epsilon, a, aa, \dots\}$. To reason about complex diagrams, it is easier to assign variable names to each wire: let us call $N$ to the top wire of the feedback loop, and $M$ to the middle wire joining $\overset{\rightarrow}{\rightarrow}\!\!\bullet\!\!\vdash$ to $\rightarrow\!\!\bullet\overset{\curvearrowright}{\phantom{a}}$. After simplifying a few redundant constraints, we get

$$\left[\!\!\left[\; \begin{matrix} N \\ \vphantom{M} \\ L \end{matrix} \;\right]\!\!\right] \begin{aligned} &= \{(L,K) \mid \exists M.N.\; L, N \subseteq M,\; Ma \subseteq N, M \subseteq K\} \\ &= \{(L,K) \mid \exists M.\; L \subseteq M,\; Ma \subseteq M\, M \subseteq K\} \\ &= \{(L,K) \mid \exists M.\; L \cup Ma \subseteq M\, M \subseteq K\}. \end{aligned}$$

Call this diagram $d$. By Arden's lemma [2], $La^*$ is the *smallest* solution of the language inequality $L \cup Ma \subseteq M$; thus $\exists M.\ L \cup Ma \subseteq M \Leftrightarrow \exists M.\ La^* \subseteq M$ and

$$\llbracket d \rrbracket = \{(L,K) \mid \exists M.\ La^* \subseteq M,\ M \subseteq K\} = \{(L,K) \mid La^* \subseteq K\}.$$

## 3. Inequational theory

In Figure 1 we introduce KDA, the theory of *Kleene Diagram Algebra*, on Syn. Once we have shown how to encode automata into it, we will show that it is *complete* for equivalence of automata-diagrams (Definition 3.1 below). We explain some salient features of KDA below. As explained in Appendix A, we use equality as a shorthand for two inequalities.

- (A1)-(A2) relate ⌣̇ and ⌣, allowing us to bend and straighten wires at will. This makes Syn modulo (A1)-(A2), a *compact closed category* [27]. (A3) allows us to eliminate isolated loops.

- The B block states that →•⌣, →• forms a cocommutative comonoid (B1)-(B3), while ⌣•→, •→ form a commutative monoid (B4)-(B6). Moreover, →•⌣, →•, ⌣•→, •→ together form an idempotent bimonoid (B7)-(B11). Incidentally, note that (B1)-(B11) axiomatise the SMC of finite sets and relations, with monoidal product given by the disjoint sum. (B12) allows us to eliminate trivial feedback loops, extending the previous axiomatisation to the traced SMC of sets and relations.

- The C block makes (⌣○→, ○→) into a commutative monoid and (→○⌣, →○) into a cocommutative comonoid.

- The D block states that both (→•⌣, →•, ⌣○→, ○→) and (→○⌣, →○, ⌣•→, •→) form two bimonoids.

- The E block encodes fundamental inequations of the theory of Cartesian bicategories [14]. They are lax versions of distributive laws, of ⌣○→ over ⌣•→ and →○⌣ over →•⌣ (as well as their units and counits). We also have equalities that force the —$\boxed{a}$— to distribute over the other operations. Semantically, this corresponds to the fact that the —$\boxed{a}$— are lattice homomorphism.

- The F block state a number of adjunctions in the 2-categorical sense[1]: two morphisms $f : X \to Y$ and $g : Y \to X$ are adjoint if $id_X \leq f\,\text{;}\,g$ and $g\,\text{;}\,f \leq id_Y$. We write $f \vdash g$ and say that $f$ is left ajdoint to $g$. The situation for KDA is summarised by the following six adjunctions:

$$\text{⌣○→} \dashv \text{→•⌣} \dashv \text{⌣•→} \dashv \text{→○⌣} \tag{3.1}$$

$$\text{○→} \dashv \text{→•} \dashv \text{•→} \dashv \text{→○} \tag{3.2}$$

The central adjunctions involving only →•⌣, →•, ⌣•→, •→ are the key defining adjunctions of Cartesian bicategories. The remaining adjunctions hold whenever the supporting poset is a lattice (has binary meets and joins), which is the case for the set of (regular) languages over a given alphabet, as it is closed under union and intersection. To better understand where these adjunctions come from, it is helpful to adopt a semantic point of view. For example, recall that $\llbracket \text{→○⌣} \rrbracket = \{(L,(K_1,K_2)) \mid L \subseteq K_1 \cup K_2\}$ and

---

[1] In this setting, the 2-cells are simply inclusions, so the reader can also think about these adjunctions simply as Galois connections.
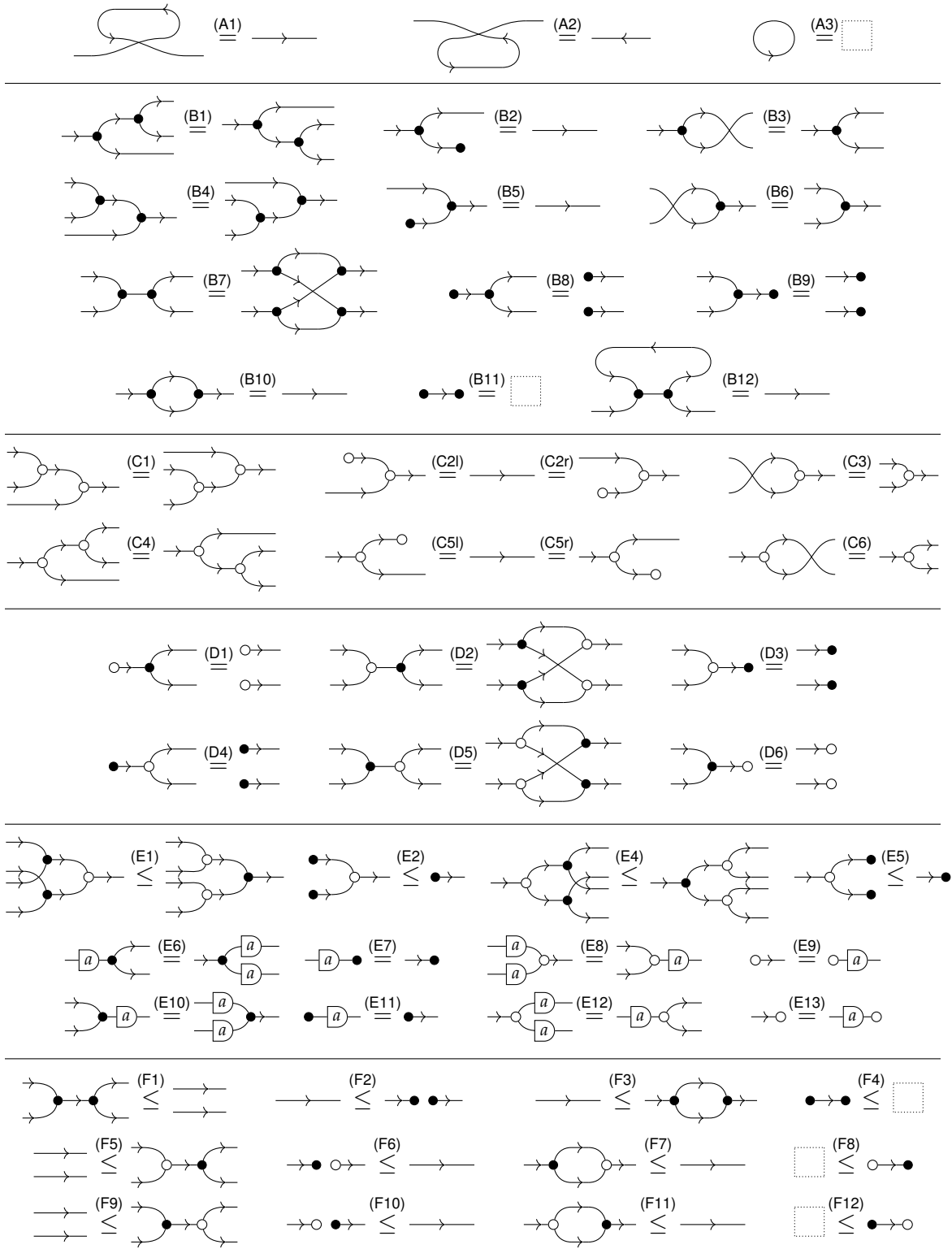
Figure 1: Theory of Kleene Diagram Algebra (KDA).

$\left[\!\!\left[ \to\!\bullet\!\overset{\rightarrow}{\underset{\rightarrow}{\frown}} \right]\!\!\right] = \left\{ \left( L, (K_1, K_2) \right) \mid L \subseteq K_i,\, i = 1, 2 \right\} = \left\{ \left( L, (K_1, K_2) \right) \mid L \subseteq K_1 \cap K_2 \right\}$. Thus, one can see the adjunction $\overset{\rightarrow}{\underset{\rightarrow}{\circ}}\!\!\succ \dashv \to\!\bullet\!\overset{\rightarrow}{\underset{\rightarrow}{\frown}}$ as arising from the duality between the two different ways of turning intersection – a monotone map – into a monotone relation, *cf.* Remark 2.3. These two embeddings of the same monotone map always give rise to an adjunction of this form. Note that we can strengthen some of the inequalities in this block to equalities: the equalities for (F6),(F8),(F10), and (F12) can all be derived.

- The equational theory contains a number of important symmetries: many equations also hold when taking the horizontal reflection of the diagrams involved, e.g., (B1) and (B4) or (D2) and (D5). This will play an important role in our proof of completeness in Section 5.
- Finally, the proposed axiomatisation is *not* minimal. For example (F3) and (F4) are obviously subsumed by (B10) and (B11). Similarly, the equations of the E block could be weakened to inequalities. As we will frequently make use of several symmetries of the equational theory, for the reader's convenience we have preferred to add these redundant axioms to Fig. 1, instead of scattering them in different subsequent lemmas. They also serve to highlight common algebraic structures that occur in related theories (*e.g.* bimonoids).

From the data of the set of generators and the relations of Fig. 1, we can construct a partial order on each homset of Syn as explained in Appendix A.2. First we build a preorder on each homset by closing KDA under $\oplus$ and taking the reflexive and transitive closure of the resulting relation. Then we obtain a partial order by quotienting the resulting pre-order to impose anti-symmetry. Below, we will call $\leq_{KDA}$ the resulting order on each homset.

We are interested in the properties of those diagrams that correspond to automata. As we will see in the next section, there is a close relationship between diagrams composed exclusively of the generators in (2.1) and standard automata, justifying the following definition.

**Definition 3.1.** An *automaton-diagram* is a morphism of Syn built from the generators of (2.1), namely

$$\to\!\bullet\!\overset{\rightarrow}{\underset{\rightarrow}{\frown}},\ \to\!\bullet,\ \overset{\rightarrow}{\underset{\rightarrow}{\circ}}\!\!\succ,\ \bullet\!\!\succ,\ -\!\boxed{a}\!-,\ \overset{\rightarrow}{\supset},\ \subsetneq.$$

We call $\mathrm{Aut}_\Sigma$ the corresponding monoidal subcategory.

**Remark 3.2.** Note that, as anticipated in the introduction, constructing automata does not require the white generators of Syn, as defined in (2.2). However, together with the corresponding adjoint structure in the inequational theory, they are essential to the completeness proof given below.

We can now state our soundness and completeness result for automata-diagrams.

**Theorem 3.3** (Soundness and Completeness)**.** *For any two automata-diagrams $d$ and $d'$,*

$$[\![d]\!] \subseteq [\![d']\!]\ \ \textit{if and only if } d \leq_{KDA} d'.$$

The *soundness* of $\leq_{KDA}$ for the chosen interpretation is not difficult to show and involves a routine verification that all the axioms in Fig. 1 hold in the semantics. We show (D2) here as an example. We have

$$\left[\!\!\left[ \overset{\rightarrow}{\underset{\rightarrow}{\circ}}\!\!-\!\!\circ\!-\!\bullet\!\overset{\rightarrow}{\underset{\rightarrow}{\frown}} \right]\!\!\right] = \{((L_1, L_2), (K_1, K_2)) \mid \exists M.\ L_1 \cap L_2 \subseteq M \subseteq K_1 \cap K_2\}$$

$$= \{((L_1, L_2), (K_1, K_2)) \mid L_1 \cap L_2 \subseteq K_1 \cap K_2\}$$

$$\subseteq \left\{ ((L_1, L_2), (K_1, K_2)) \mid \begin{array}{l} L_1 \subseteq L_1 \cup (K_1 \cap K_2), \ L_2 \subseteq L_2 \cup (K_1 \cap K_2), \\ L_1 \cap L_2 \subseteq K_1, \ (K_1 \cap K_2) \cap (K_1 \cap K_2) \subseteq K_2, \end{array} \right\}$$

$$\subseteq \left\{ ((L_1, L_2), (K_1, K_2)) \mid \exists M_1.M_2.M_3.M_4. \begin{array}{l} L_1 \subseteq M_1 \cup M_3, \\ L_2 \subseteq M_2 \cup M_4, \\ M_1 \cap M_2 \subseteq K_1, \\ M_3 \cap M_4 \subseteq K_2 \end{array} \right\}$$

$$= \left[\!\!\left[ \ \rightbackground \ \right]\!\!\right]$$

and, conversely

$$\left[\!\!\left[ \ \rightbackground \ \right]\!\!\right] = \left\{ ((L_1, L_2), (K_1, K_2)) \mid \exists M_1.M_2.M_3.M_4. \begin{array}{l} L_1 \subseteq M_1 \cup M_3, \\ L_2 \subseteq M_2 \cup M_4, \\ M_1 \cap M_2 \subseteq K_1, \\ M_3 \cap M_4 \subseteq K_2 \end{array} \right\}$$

$$\subseteq \left\{ ((L_1, L_2), (K_1, K_2)) \mid \exists M_1.M_2.M_3.M_4. \begin{array}{l} L_1 \cap L_2 \\ \subseteq (M_1 \cup M_3) \cap (M_2 \cup M_4) \\ \subseteq (M_1 \cap M_2) \cup (M_3 \cap M_4) \\ \subseteq K_1 \cap K_2 \end{array} \right\}$$

$$\subseteq \left\{ ((L_1, L_2), (K_1, K_2)) \mid \exists M. \ L_1 \cap L_2 \subseteq M \subseteq K_1 \cap K_2 \right\}$$

$$= \left[\!\!\left[ \ \rightbackground \ \right]\!\!\right].$$

**Remark 3.4.** The two black generating objects are not *discrete* in the terminology of Carboni and Walters [14]: this means that $\rightarrow\!\!\bullet\!\!\text{<}$, $\rightarrow\!\!\bullet$ and $\text{>}\!\!\bullet\!\!\rightarrow$, $\bullet\!\!\rightarrow$ do not satisfy the Frobenius law. In fact, because they already form a bimonoid, satisfying the Frobenius law would trivialise the equational theory, making any two diagrams of the same type equal.

**Remark 3.5.** The atomic actions $-\!\boxed{a}\!-$ ($a \in \Sigma$) compose freely. This is because, as we study automata, we are interested in the *free* monoid $\Sigma^*$ over $\Sigma$. However, nothing would prevent us from modelling other structures. Free commutative monoids (powers of $\mathbb{N}$), whose rational subsets correspond to semilinear sets [16, Chapter 11] would be of particular interest.

**Remark 3.6.** Semantically, the generators of automata-diagrams allow us to specify systems of linear language inequalities of the form $La \subseteq K$. The addition of the white nodes extends the expressiveness of our calculus, giving us the ability to specify systems involving intersection and union on both sides of an inequality. While it is clear that this is a strictly more expressive calculus – for example, the relations that interpret any of the white generators cannot be expressed using only the black ones – we leave the precise characterisation of the image of $[\![\cdot]\!]$ for future work.

**Remark 3.7.** We have already explained that the adjunctions between the white and black nodes hold whenever the underlying poset is a lattice. In fact, the calculus we give in this paper could be interpreted over an arbitrary lattice, with the semantics of each $-\!\boxed{a}\!-$ ($a \in \Sigma$) given as a lattice endomorphism.

This last claim also shows that KDA is incomplete for the given interpretation. Indeed, the lattice of languages (and that of regular languages) is a Boolean algebra. This additional structur can be captured equationally, by making $(\to\!\!\overrightarrow{\subset}, \to\!\circ, \overrightarrow{\supset}\!\!\leftarrow, \circ\!\leftarrow)$ a Frobenius algebra. However, the defining equations of Frobenius algebras cannot be derived from KDA. One way to show this is to devise a counter-model by interpreting the calculus over a different lattice, which is not complemented.

Note that this is a feature, not a bug. We are hoping that the methods of this paper can be translated to other settings, *e.g.* to automata-like structures or modal logics that do not require the underlying semantics to be Boolean.

We will now write $\leq_{KDA}$ (resp. $=_{KDA}$) simply as $\leq$ (resp. $=$) to simplify notation, and say that diagrams $c$ and $d$ of the same type are *equal* when $c =_{KDA} d$.

## 4. ENCODING REGULAR EXPRESSIONS AND AUTOMATA

A major appeal of our approach is that both regular expressions and automata can be uniformly represented in the graphical language of string diagrams, and the translation of one into the other becomes a (in)equational derivation in KDA. In fact, we will see there is a close resemblance between automata and the shape of the string diagrams interpreting them—the main difference being that string diagrams can be composed.

In this section we describe how regular expressions (resp. automata) can be encoded as string diagrams, such that their semantics corresponds in a precise way to the languages that they describe (resp. recognise).

4.1. **From regular expressions to string diagrams.** We can define an encoding $\langle - \rangle$ of regular expressions into string diagrams of $\mathsf{Aut}_\Sigma$ inductively as follows:



(4.1)

For example,



(4.2)

Let $[\![e]\!]_R \in \mathcal{L}_\Sigma$ be the standard semantics of a regular expression $e$, defined inductively as follows:

$$[\![e+f]\!]_R = [\![e]\!]_R \cup [\![f]\!]_R \quad [\![ef]\!]_R = \{vw \mid v \in [\![e]\!]_R, w \in [\![f]\!]_R\}$$

$$[\![1]\!]_R = \{\varepsilon\} \qquad [\![0]\!]_R = \varnothing \qquad [\![a]\!]_R = \{a\}\,(a \in \Sigma) \qquad [\![e^*]\!]_R = \bigcup_{n \in \mathbb{N}} [\![e^n]\!]_R$$

where $e^{n+1} := ee^n$ and $e^0 := 1$. As expected, the translation preserves the language interpretation of regular expressions in a sense that the following proposition makes precise.

**Proposition 4.1.** *For $e, f$ two regular expressions, $\llbracket e \rrbracket_R = \llbracket f \rrbracket_R$ iff $\llbracket \langle e \rangle \rrbracket = \llbracket \langle f \rangle \rrbracket$.*

*Proof.* To prove the statement, it is enough to show that $\llbracket \langle e \rangle \rrbracket = \{(L, K) \mid L \llbracket e \rrbracket_R \subseteq K\}$. We do so by induction on the structure of regular expressions. Note that we write "$;$" for relational composition, from left to right: $R \, ; \, S = \{(x, z) \mid \exists y, (x, y) \in R, (y, z) \in S\}$.

The proposition holds by definition for the generators: $\llbracket \langle a \rangle \rrbracket = \{(L, K) \mid La \subseteq K\}$. There are three inductive cases to consider. Assume that $e$ and $f$ satisfy the proposition.

- For the $ef$ case, $\llbracket \langle ef \rangle \rrbracket = \llbracket \langle e \rangle \rrbracket \, ; \, \llbracket \langle f \rangle \rrbracket = \{(L, K) \mid L \llbracket e \rrbracket_R \subseteq K\} \, ; \, \{(L, K) \mid L \llbracket f \rrbracket_R \subseteq K\}$. Hence, by monotony of the product, we have $\llbracket \langle ef \rangle \rrbracket = \{(L, K) \mid L \llbracket e \rrbracket_R \llbracket f \rrbracket_R \subseteq K\} = \{(L, K) \mid \llbracket ef \rrbracket_R L \subseteq K\}$.

- For the case of $e + f$ we have

$$\llbracket \langle e + f \rangle \rrbracket = \left\{ (L, K) \,\middle|\, \exists K_1, K_2, L_1, L_2. \begin{array}{l} K_1, K_2 \subseteq K, \\ L \subseteq L_1, L_2, \\ L_1 \llbracket e \rrbracket_R \subseteq K_1, \\ L_2 \llbracket f \rrbracket_R \subseteq K_2 \end{array} \right\}$$

$$= \left\{ (L, K) \,\middle|\, \exists L_1, L_2. \begin{array}{l} L \subseteq L_1, L_2, \\ L_1 \llbracket e \rrbracket_R \subseteq K, \\ L_2 \llbracket f \rrbracket_R \subseteq K \end{array} \right\}$$

$$= \left\{ (L, K) \,\middle|\, \exists L_1, L_2. \begin{array}{l} L \subseteq L_1, L_2, \\ L_1 \llbracket e \rrbracket_R \cup L_2 \llbracket f \rrbracket_R \subseteq K \end{array} \right\}$$

$$= \{(L, K) \mid L \llbracket e \rrbracket_R \cup L \llbracket f \rrbracket_R \subseteq K\}$$

$$= \{(L, K) \mid L(\llbracket e \rrbracket_R \cup \llbracket f \rrbracket_R) \subseteq K\}$$

$$= \{(L, K) \mid L \llbracket e + f \rrbracket_R \subseteq K\}$$

- Finally, for $e^*$,

$$\llbracket \langle e^* \rangle \rrbracket = \{(L, K) \mid \exists M, N. \ M, L \subseteq N, N \llbracket e \rrbracket_R \subseteq M, N \subseteq K\}$$

$$= \{(L, K) \mid \exists N. \ N \llbracket e \rrbracket_R \subseteq N, L \subseteq N \subseteq K\}$$

$$= \{(L, K) \mid \exists N. \ L \cup N \llbracket e \rrbracket_R \subseteq N, L \subseteq N \subseteq K\}$$

$$\overset{(\star)}{=} \{(L, K) \mid \exists N. \ L \llbracket e \rrbracket_R^* \subseteq N, L \subseteq N \subseteq K\}$$

$$= \{(L, K) \mid \exists N. \ L \llbracket e^* \rrbracket_R \subseteq N, L \subseteq N \subseteq K\}$$

$$= \{(L, K) \mid L \llbracket e^* \rrbracket_R \subseteq K\}$$

where the starred equation is a consequence of Arden's lemma [2]: $A^* B$ is the smallest solution (for $X$) of the language equation $B \cup AX \subseteq X$, where we write $A^*$ for the language $\bigcup_{n \geq 0} A^n$.

$\square$

**Remark 4.2.** Regular expressions can also be interpreted as binary relations over an arbitrary set: such an interpretation is given by a mapping of each letter to a binary relation on some set, and extended inductively to all regexes (with the sum interpreted as union, product as relational composition, 1 as the identity, 0 as the empty relation, and the star as the reflexive, transitive closure). We can write $\mathsf{Rel} \vDash e = f$ if every relational interpretation identifies $e$ and $f$. The statement and proof of Proposition 4.1 follow a slight modification of the proof that, $\mathsf{Rel} \vDash e = f$ implies $\llbracket e \rrbracket_R = \llbracket f \rrbracket_R$[2]. The standard argument proceeds as follows:

---

[2] We thank the anonymous reviewer for pointing this out. The earliest reference we could find is [40, p.24]

define a map $\sigma : \Sigma \to 2^{\Sigma^\star \times \Sigma^\star}$ given by $\sigma(a) = \{(w, wa) \mid w \in \Sigma^\star\}$, which can be extended inductively to a map $\hat{\sigma}$ defined over all regexes. Then, one can show by induction that $\hat{\sigma}(e) = \{(w, wu) \mid u \in \llbracket e \rrbracket_R\}$ so that, in particular, $\llbracket e \rrbracket_R = \{w \mid (\epsilon, w) \in \hat{\sigma}(e)\}$. The version presented above modifies this idea by adding inclusion where necessary, to turn all the relevant relations into *monotone* relations.

From a diagrammatic perspective, regular expressions correspond to diagrams that enforce a restricted form of composition. They can be characterised in the syntax as the image of $\langle \cdot \rangle$ or, equivalently, as those diagrams of type $\blacktriangleright \to \blacktriangleright$ built inductively from the following three operations
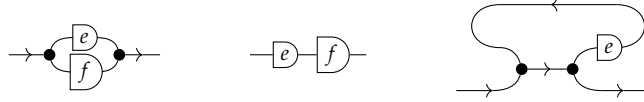


starting from the basic diagrams $—\boxed{a}—$, $—\bullet\ \bullet—$, and $——\rightarrow—$. In what follows, we will refer to any diagram of this form as a *regex-diagram*.

4.2. **From automata to string diagrams.** Example (4.2) suggests that the string diagram $\langle e \rangle$ corresponding to a regular expression $e$ looks a lot like a nondeterministic finite-state automaton (NFA) for $e$. In fact, the translation $\langle - \rangle$ can be seen as the diagrammatic counterpart of Thompson's construction [46] that builds an NFA from a regular expression.

We can generalise the encoding of regular expressions and translate NFA directly into string diagrams, in at least two ways. The first is to encode an NFA as the diagrammatic counterpart of its transition relation. The second is to translate directly its graph representation into the diagrammatic syntax.

**Encoding the transition relation.** This is a simple variant of the translation of matrices over semirings that has appeared in several places in the literature [33, 48].

Let $A$ be an NFA with set of states $Q$, initial state $q_0 \in Q$, accepting states $F \subseteq Q$ and transition relation $\delta \subseteq Q \times \Sigma \times Q$. We can represent $\delta$ as a string diagram $d$ with $|Q|$ incoming wires on the left and $|Q|$ outgoing wires on the right.The left $j$-th port of $d$ is connected to the $i$-th port on the right through an $—\boxed{a}—$ whenever $(q_i, a, q_j) \in \delta$. To accommodate nondeterminism, when the same two ports are connected by several different letters of $\Sigma$, we join these using $\rightarrow\!\!\bullet\!\!\overset{\curvearrowleft}{\curvearrowleft}$ and $\overset{\curvearrowright}{\rightarrow}\!\!\bullet\!\!\rightarrow$ . When $(q_i, \epsilon, q_j) \in \delta$, the two ports are simply connected via a plain identity wire. If there is no tuple in $\delta$ such that $(q_i, a, q_j) \in \delta$ for any $a$, the two corresponding ports are disconnected, using $\rightarrow\!\!\bullet\ \bullet\!\!\rightarrow$ if necessary.

For example, the transition relation of an NFA with three states and $\delta = \{((q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_1), (q_2, a, q_2))\}$ (disregarding the initial and accepting states for the moment) is depicted on the right. Conversely, given such a diagram, we can recover $\delta$ by collecting $\Sigma$-weighted paths from left to right ports.

$$d = \quad$$
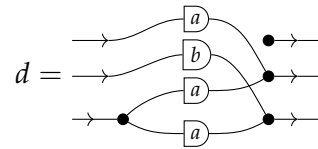


To deal with the initial state, we add an additional incoming wire connected to the right port corresponding to the initial state of the automaton. Similarly, for accepting states we add an additional outgoing wire, connected to the left ports corresponding to each accepting state, via $\overset{\curvearrowright}{\rightarrow}\!\!\bullet\!\!\rightarrow$ if there is more than one.

Finally, we trace out the $|Q|$ wires of the diagrammatic transition relation to obtain the associated string diagram. In other words, for a NFA with initial state $q_0$, set of accepting states $F$, transition relation $\delta$, we obtain the string diagram on the right, where $d$ is the diagrammatic counterpart of $\delta$ as defined above, $e$ is the injection of a single wire as the first amongst $|Q|$ wires, and $f$ discards all wires that are not associated to states in $F$ with $\rightarrow\bullet$, and applies $\rightarrow\!\!\bullet\!\!\rightarrow$ to merge them into a single outgoing wire.
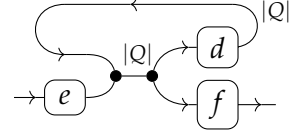
For example, if $A$ with $\delta$ as above has initial state $q_0$ and set of accepting states $\{q_2\}$, we get the diagram below left; if instead, all states are accepting, we obtain the diagram below right:
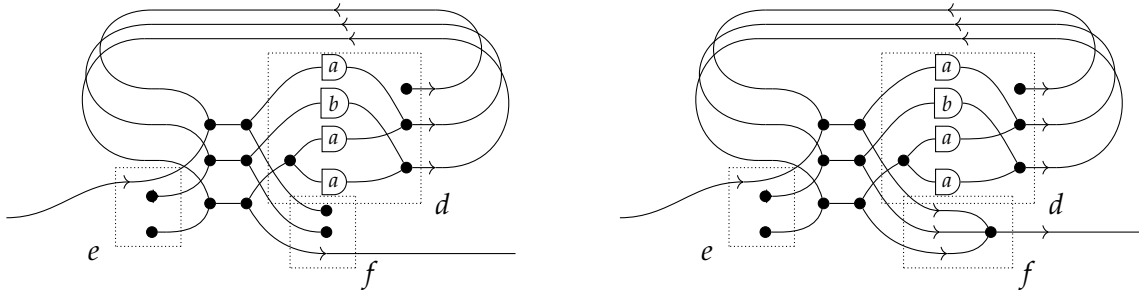
The correctness of this simple translation is justified by a semantic correspondence between the language recognised by a given NFA $A$ and the denotation of the corresponding string diagram.

**Proposition 4.3.** *Given an NFA A which recognises the language L, let $c_A$ be its associated string diagram, constructed as above. Then $[\![c_A]\!] = \{(K, K') \mid LK \subseteq K'\}$.*

*Proof.* This is the diagrammatic counterpart of the representation of automata as matrices of regular expressions given in [29, Definition 12].

We write **K** for a vector of languages $(K_1, \ldots, K_{|Q|})$ and, for a square matrix of languages **A**, let **AK** be the language vector resulting from applying **A** to **K** in the obvious way (note that we use standard matrix multiplication order, which is the opposite of the diagrammatic order). By [29, Theorem 11], square language matrices form a Kleene algebra, with the composition of matrices as product, component-wise union as sum and the star defined as in [29, Lemma 10]. We also write write $\mathbf{K} \subseteq \mathbf{K}'$ if the inclusions all hold component-wise. Furthermore, Arden's lemma holds in this slightly more general setting: the least solution of the language-matrix equation $\mathbf{B} \cup \mathbf{AX} \subseteq \mathbf{X}$ is $\mathbf{X} = \mathbf{A}^*\mathbf{B}$. This is another consequence of the fact that matrices of languages also form a Kleene algebra [29, Theorem 11].

Now, for a given automaton $A$ we construct the diagram below as explained above:

$$c_A \;=\;$$
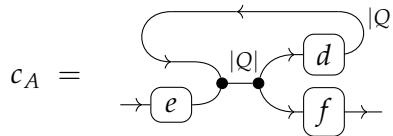
with $d$ the diagram encoding the transition relation of $A$, $e_0$ the diagram encoding its initial state, and $f$ the diagram encoding its set of final states. Let $[\![d]\!] = \mathbf{D}$ be the language matrix obtained from $A$ by letting $\mathbf{D}_{ij} = \{a\}$ if $(q_i, a, q_j)$ is in the transition relation of $A$. We

proceed as in Example 2.5. First, we have

$$
\left[\!\left[\ \vcenter{\hbox{\includegraphics{diagram}}}\ \right]\!\right] \begin{aligned} &= \{(\mathbf{K}, \mathbf{K}') \mid \exists \mathbf{M}, \mathbf{N},\ \mathbf{M}, \mathbf{K} \subseteq \mathbf{N},\ \mathbf{DN} \subseteq \mathbf{M},\ \mathbf{N} \subseteq \mathbf{K}'\} \\ &= \{(\mathbf{K}, \mathbf{K}') \mid \exists \mathbf{N},\ \mathbf{DN} \subseteq \mathbf{N},\ \mathbf{K} \subseteq \mathbf{N} \subseteq \mathbf{K}'\} \\ &= \{(\mathbf{K}, \mathbf{K}') \mid \exists \mathbf{N},\ \mathbf{K} \cup \mathbf{DN} \subseteq \mathbf{N},\ \mathbf{N} \subseteq \mathbf{K}'\} \\ &\overset{(\star)}{=} \{(\mathbf{K}, \mathbf{K}') \mid \exists \mathbf{N},\ \mathbf{D}^*\mathbf{K} \subseteq \mathbf{N}, \mathbf{N} \subseteq \mathbf{K}'\} \\ &= \{(\mathbf{K}, \mathbf{K}') \mid \mathbf{D}^*\mathbf{K} \subseteq \mathbf{K}'\} \end{aligned}
$$

where the starred step holds by the matrix-form of Arden's lemma. Then, $[\![e]\!]$ and $[\![f]\!]$ pick out the component languages of $\mathbf{D}^*$ that correspond to the initial state of $A$ and each final state, and takes their union. Thus, we get

$$
[\![c_A]\!] = \left[\!\left[\ \vcenter{\hbox{\includegraphics{diagram}}}\ \right]\!\right] \begin{aligned} &= [\![e]\!]\,;\, \{(\mathbf{K}, \mathbf{K}') \mid \mathbf{D}^*\mathbf{K} \subseteq \mathbf{K}'\}\,;\, [\![f]\!] \\ &= \{(K, K') \mid LK \subseteq K'\} \end{aligned}
$$

where $L$ is the language accepted by the original automaton.

$\square$

**From graphs to string diagrams.** The second way of translating automata into string diagrams mimics more directly the usual representation of automata as graphs. The idea (which should be sufficiently intuitive to not need to be made formal here) is, for each state, to use $\vcenter{\hbox{\includegraphics{incoming}}}$ to represent incoming edges, and $\vcenter{\hbox{\includegraphics{outgoing}}}$ to represent outgoing edges. As above, labels $a \in A$ will be modelled using $-\boxed{a}-$. For example, the graph and the associated string diagram corresponding with the NFA above are



(4.3)

Note that the initial state (which we indicate with an arrow pointing down and into a state) of the automaton corresponds to the left interface of the string diagram, and the accepting state (which we indicate with an arrow pointing down and out of a state) to the right interface of the same diagram. As before, when there are multiple accepting states, they all connect to a single right interface, via $\vcenter{\hbox{\includegraphics{connect}}}$. For example, if we make all states accepting in the automaton above, we get the following diagrammatic representation:

4.3. **From string diagrams to automata.** The previous discussion shows how NFAs can be seen as string diagrams of type $\blacktriangleright \to \blacktriangleright$. The converse is also true: we now show how to extract an automaton from any automaton-diagram $d \colon \blacktriangleright \to \blacktriangleright$, such that the language the automaton recognises matches the semantics of $d$.

In order to phrase this correspondence formally, we need to introduce some terminology. We call *left-to-right* those automata-diagrams whose domain and co-domain contain only $\blacktriangleright$, i.e. their type is of the form $\blacktriangleright^n \to \blacktriangleright^m$. The idea is that, in any such string diagram, the $n$ left interfaces act as *inputs* of the computation, and the $m$ right interfaces act as *outputs*. For instance, (4.3) is a left-to-right diagram $\blacktriangleright \to \blacktriangleright$.

We call *block* of a certain subset of generators a diagram composed only of these generators (using both $\,\fatsemi\,$ and $\oplus$), possibly including some permutation of the wires.

**Definition 4.4.** A *matrix-diagram* is a left-to-right diagram that factors as a composition of a block of $\to\!\!\bullet\!\overset{\frown}{\underset{\smile}{\phantom{.}}}, \to\!\!\bullet$, followed by a block of $-\boxed{a}-$ for $a \in \Sigma$ and finally, a block of $\overset{\frown}{\underset{\smile}{\phantom{.}}}\!\!\bullet\!\succ, \bullet\!\succ$, such that any path from a left port to a right port passes through *at most one* $-\boxed{a}-$.

To each matrix-diagram $d$ we can associate a unique transition relation $\delta$ by gathering paths from each input to each output: $(q_i, a, q_j) \in \delta$ if there is $-\boxed{a}-$ joining the $i$-th input to the $j$-th output. A transition relation is *$\epsilon$-free* if it does not contain the empty word. It is *deterministic* if it is $\epsilon$-free and, for each $i$ and each $a \in \Sigma$ there is at most one $j$ such that $(q_i, a, q_j) \in \delta$. We will apply these terms to matrix-diagrams and the associated transition relation interchangeably. The example of Section 4.2 below, with the three blocks highlighted, is a matrix-diagram.
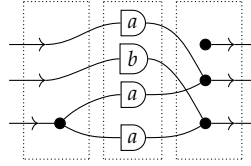


It is $\epsilon$-free but not deterministic since there are two $a$-labelled transitions starting from the third input.

We also call *relation-diagram* a matrix-diagram that contain no $-\boxed{a}-$. Intuitively, in the absence of the $-\boxed{a}-$ generators, the corresponding theory is simply that of Boolean matrices, *i.e.* relations.

**Definition 4.5** (Representation). For a diagram[3] $c \colon \blacktriangleright \to \blacktriangleright$, a *representation* is a triple $(d, e, f)$ of an $\epsilon$-free matrix-diagram $d \colon \blacktriangleright^l \to \blacktriangleright^l$ representing the transition dynamics, and two relation-diagrams $e \colon \blacktriangleright \to \blacktriangleright^l$, and $f \colon \blacktriangleright^l \to \blacktriangleright$ representing the initial and the final states respectively, such that



It is a *deterministic representation* if moreover $d$ is a deterministic matrix-diagram.

For example, given the string diagram below on the left, we can use the axioms of KDA to rewrite it to an equivalent diagram from which a representation can easily be read—the

---

[3]Representations could also be defined for arbitrary left-to-right diagrams $\blacktriangleright^m \to \blacktriangleright^n$, but we will only need them to connect diagrams and automata, so it is sufficient to consider the $n = m = 1$ case for our purpose.

highlighted matrix-diagram corresponds to the same transition matrix $d$ as in the example above:



(4.4)
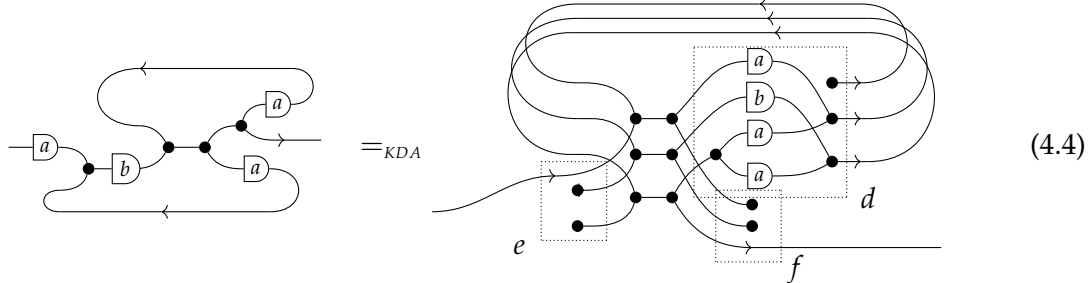
From a $\blacktriangleright\to\blacktriangleright$ diagram with representation $(d, e, f)$, we can construct an NFA as follows:

- its state set is $Q = \{q_1, \dots, q_l\}$, i.e., there is one state for each wire of $d : \blacktriangleright^l \to \blacktriangleright^l$;
- its transition relation built from $d$ as described above;
- its initial state is the only non-zero coefficient of $e : \blacktriangleright \to \blacktriangleright^l$, i.e., the only wire in the codomain of $e$ connected to the single wire in the domain);
- its final states $F$ are those $q_j$ for which the $j$-th coefficient of $f : \blacktriangleright^l \to \blacktriangleright$ is non-zero, i.e., the wires of the domain of $f$ connected to its single codomain wire.

The construction above is the inverse of that of Section 4.2. The link between the constructed automaton and the original string diagram is summarised in the following statement, which is a straightforward corollary of Proposition 4.3.

**Proposition 4.6.** *For a diagram $d : \blacktriangleright\to\blacktriangleright$ with a representation $\hat{d}$, let $A_{\hat{d}}$ be the associated automaton, constructed as above. Then $\hat{L}$ is the language recognised by $A_{\hat{d}}$ iff $[\![d]\!] = \{(K, K') \mid \hat{L}K \subseteq K'\}$.*

The next proposition is crucial: it states that a representation can be extracted from any diagram $\blacktriangleright\to\blacktriangleright$.

**Proposition 4.7.** *Any automaton-diagram $\blacktriangleright\to\blacktriangleright$ has a representation.*

We will need to prove a few preliminary results before tackling the proof of Proposition 4.7. The following lemma will also be needed in the determinisation procedure of Section 5.3.

The next theorem establishes completeness for a restricted fragment of our language, corresponding to matrices of finite languages. More precisely, every diagram formed only of the generators $\to\!\!\!\overset{\rightharpoonup}{\bullet_{\rightharpoonup}}, \to\!\!\bullet, \overset{\rightharpoonup}{\bullet}\!\!\succ, \bullet\!\!\succ, -\boxed{a}-$ is interpreted via $[\![\cdot]\!]$ as a matrix with coefficients in $\mathbb{B}(\Sigma^*)$, the semiring of *finite* sets of words over the alphabet $\Sigma$.

**Theorem 4.8** (Matrix completeness). *For any two diagrams $c, d$ formed only of the generators $\to\!\!\!\overset{\rightharpoonup}{\bullet_{\rightharpoonup}}, \to\!\!\bullet, \overset{\rightharpoonup}{\bullet}\!\!\succ, \bullet\!\!\succ, -\boxed{a}-$, we have $[\![c]\!] = [\![d]\!]$ iff $c = d$.*

*Proof.* This result is particular case of a standard fact, that can be found more generally for any semiring in [48, Chapter 3] for example. The required equalities are axioms (B1)-(B11) and (E6-11). $\qquad\square$

The equalities (E6-7) and (E10-11) in Fig. 1 can be extended to any matrix-diagram.

**Lemma 4.9** (Matrix distributivity). *Any matrix-diagram $d : \blacktriangleright^m \to \blacktriangleright^n$ (cf. Definition 4.4) satisfies*
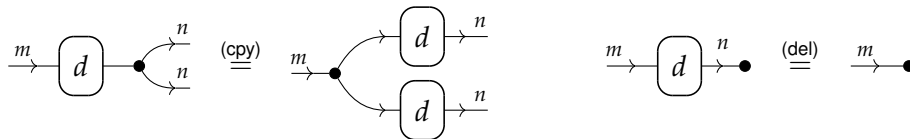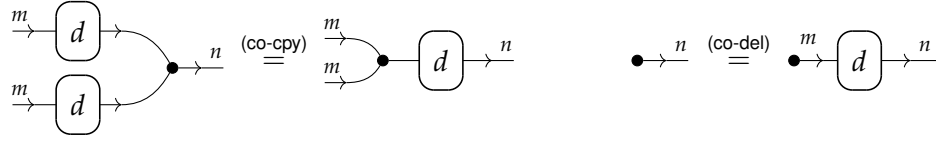
*Proof.* This lemma can be easily proved by induction, using axioms (B1)-(B11) and local distributivity (equalities (E6-7) and (E10-11)) as base case. But it is also an immediate consequence of Theorem 4.8 and of the equivalent semantic statement for matrices with coefficients in $\mathbb{B}(\Sigma^*)$. $\square$

Given a matrix-diagram $d : \blacktriangleright^{l+m} \to \blacktriangleright^{p+n}$, we will write $d_{ij}$, with $i = l, n$ and $j = p, m$, to refer to the diagram obtained from discarding all but the left $i$-ports with $\bullet\!\!\to\!\!-$ and all but the right $j$-ports with $\to\!\!\bullet$. For example,

$$d_{m,p} = \quad \begin{array}{c} l \bullet\!\!\to\!\!- \\ m \to\!\!\!\!- \end{array} \boxed{d} \begin{array}{c} \to\!\!\!- p \\ \to\!\!\bullet n \end{array}$$

The following lemma states that this operation selects the corresponding submatrices of (the matrix corresponding to) $d$.

**Lemma 4.10.** *For any matrix-diagram $d : \blacktriangleright^{l+n} \to \blacktriangleright^{p+m}$, with $d_{ij}$ defined as above. , we have*

$$d = \quad \begin{array}{c} \boxed{d_{l,p}} \\ \boxed{d_{l,n}} \\ \boxed{d_{m,p}} \\ \boxed{d_{m,n}} \end{array}$$

*Proof.* This could be proven from Lemma 4.9 but we can appeal once again to the corresponding fact for matrices over a semiring and to the completeness of our theory for matrices over $\mathbb{B}(\Sigma^*)$ (Theorem 4.8) to deduce it immediately. $\square$

Note that if we discard all but one port on the left and one port on the right, we pick out a dimension-one submatrix, i.e. a coefficient, of the corresponding matrix. Then, Lemma 4.10 is only saying that matrix-diagrams are fully characterised by their coefficients.

The following lemma establishes a useful form for diagrams.

**Lemma 4.11** (Trace canonical form). *For any automaton-diagram $c : \blacktriangleright^n \to \blacktriangleright^m$, we can always find a relation-diagram $r : \blacktriangleright^{l+n} \to \blacktriangleright^{l+m}$ such that*

$$\begin{array}{c} n \quad\boxed{c}\quad m \end{array} = \begin{array}{c} n \to \boxed{r}\;\boxed{x}\; m \end{array}$$

*where $-\!\boxed{x}\!-^l$ denotes a vertical composite of $l$-many $-\!\boxed{a}\!-$ generators.*

*Proof.* We reason by structural induction on $\mathrm{Aut}_\Sigma$. For the base case, if $c$ is $-\!\boxed{a}\!-$, we have

$$-\!\boxed{a}\!- \stackrel{(A1)}{=} \begin{array}{c} -\!\boxed{a}\!\!\smile \end{array} = \begin{array}{c} \frown\!\boxed{a} \\ \to\!\!\times \end{array}$$

and every other generator is trivially in the right form, with the trace taken over the 0 object (the empty list of generators).

There are two inductive cases to consider:

- $c$ is given by the sequential composition of two morphisms of the appropriate form (using the induction hypothesis). Then
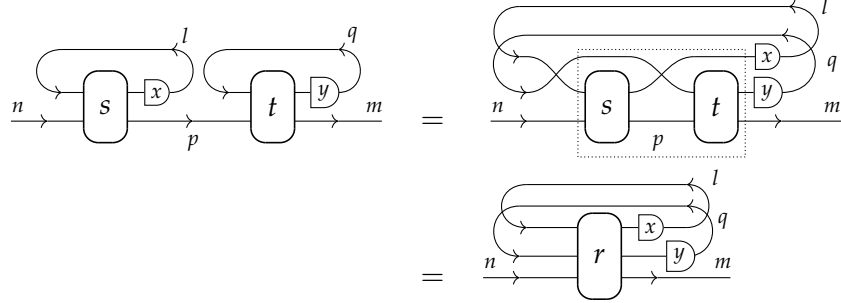


Here, the composite of the two relation diagrams $s$ and $t$ is also equal to a relation diagram, $r$, by completeness of KDA for matrices over $\mathbb{B}(\Sigma^*)$ (Theorem 4.8) so a fortiori for Boolean matrices (those $\mathbb{B}(\Sigma^*)$-matrices that contain only $0$ or $\epsilon$ coefficients).

- $c$ is given as the monoidal product of two morphisms of the appropriate form. Then
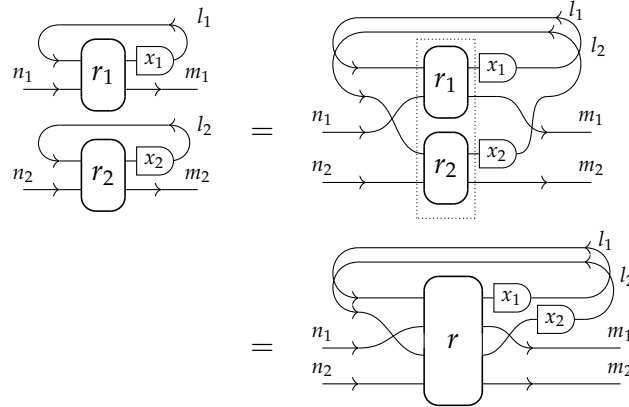


where it is immediate that $r$ is a relation diagram, as product of the two relation diagrams $r_1$ and $r_2$.

$\square$

We are now ready to prove that any automaton-diagram $c : \blacktriangleright \to \blacktriangleright$ has a representation.

*Proof of Proposition 4.7.* We first rewrite $c$ to trace canonical form (Lemma 4.11)


$$(4.5)$$

where the relation-diagram $r$ contains no $-\boxed{a}-$, and therefore factorises as a first layer of comonoid $\to\!\!\prec, \to\!\bullet$ (potentially followed by some permutations) and a third layer of vertical compositions of the monoid $\stackrel{\rightarrow}{\to}\!\!\bullet\!\succ-, \bullet\!\succ-$.

Then, we can decompose $r : \blacktriangleright^{k+1} \rightarrow \blacktriangleright^{k+1}$ as in Lemma 4.10 to obtain
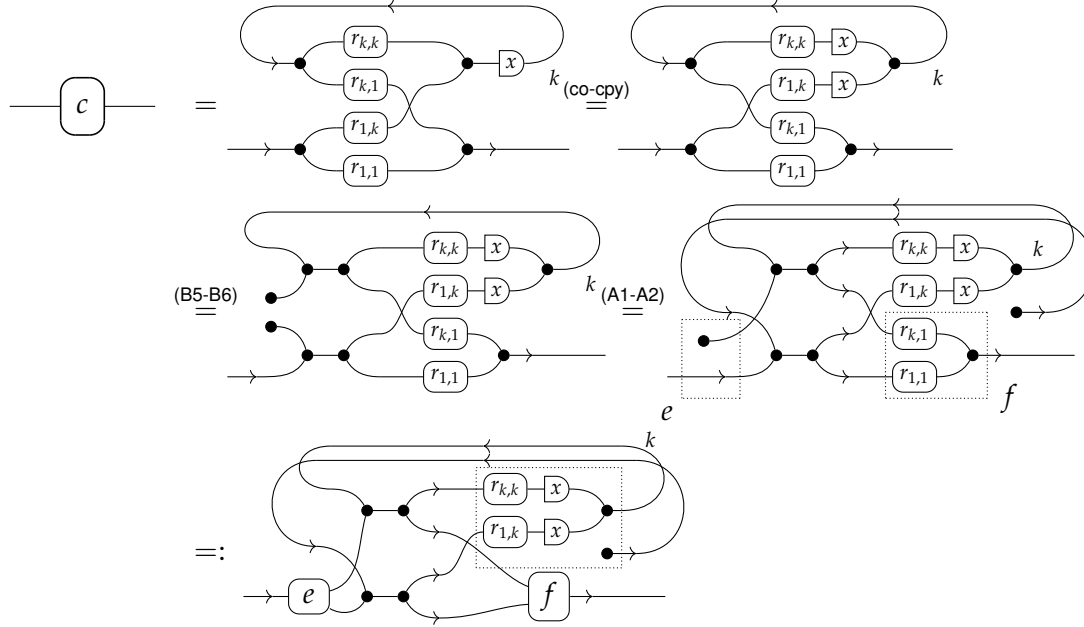


We are very close to obtaining the desired representation but the highlighted sub-diagram in the last diagram is not quite of the right form. Recall that $r_{k,k}$ and $r_{1,k}$ are relation-diagrams, which means that they factor as a block of $\rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset}, \rightarrow\!\!\bullet$ composed sequentially with a block of $\overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow, \bullet\!\!\rightarrow$. Therefore, to obtain an $\epsilon$-free matrix-diagram we can push all the scalars in $-\boxed{x}-$ into $r_{k,k}$ and $r_{1,k}$ past the $\overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow, \bullet\!\!\rightarrow$ block, using (E10)[4]. In doing so, we get two $\epsilon$-free matrix-diagrams –let us call them $d_{k,k}$ and $d_{1,k}$, respectively – and can write:
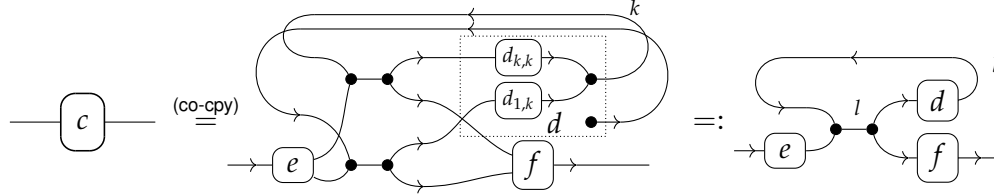


for $l = k + 1$. We can see directly from the form of this last expression that $(e, d, f)$ is a representation for $c$. $\qquad\square$

## 5. COMPLETENESS AND DETERMINISATION

This section is devoted to prove our completeness result, Theorem 3.3. We use a normal form argument: more specifically we mimic automata-theoretic results to rewrite every string diagram to a normal form corresponding to a *minimal* deterministic finite automaton (DFA). It is a standard result that, for a given regular language $L$, there is a minimal (in the number of states) DFA which recognises $L$ and that this DFA is unique up to renaming of the states. For a review of this fundamental result, we refer the reader to [31, §13-16] There are several ways to obtain a minimal DFA that is language-equivalent to a given NFA. We

---

[4]This step implements the diagrammatic counterpart of a standard $\epsilon$-elimination procedure for NFA.

will use Brzozowski's algorithm [13], which we implement in KDA itself as a sequence of diagrammatic (in)equalities. The proof proceeds in four distinct steps.

- We first show (Section 5.1) how the problem of completeness for all of $\mathrm{Aut}_\Sigma$ can be reduced to that of *equality of* $\blacktriangleright\rightarrow\blacktriangleright$ *diagrams*.
- We then give (Section 5.3) a procedure to *determinise* (the representation of) a diagram: this step consists in eliminating all subdiagrams that correspond to nondeterministic transitions in the associated automaton. For this, we build on the results of Section 5.2, in which we show that the standard subset construction can be carried out diagrammatically.
- We use the previous step to implement a *minimisation* procedure (Section 5.4) from which we obtain a minimal representation for a given diagram: this is a representation whose associated automaton is minimal—with the fewest number of states—amongst DFAs that recognise the same language. To do this, we show how the four steps of Brzozowski's minimisation algorithm (reverse; determinise; reverse; determinise) translate into diagrammatic equational reasoning. Note that the first three steps taken together simply amount to applying in reverse the determinisation procedure we have already devised. That this is possible will be a consequence of the symmetry of $\leq_{KDA}$.
- Finally, from the uniqueness of minimal DFAs, any two diagrams that have the same denotation are both equal to the same minimal representation and we can derive completeness of $\leq_{KDA}$ (Theorem 3.3).

**Remark 5.1.** At this point, it is helpful to explain the relationship between the completeness result of this paper and the erroneous claim of [38]. In the present paper, the white nodes play a double role: 1) they allow us to reduce completeness to automata-diagrams of type $\blacktriangleright\rightarrow\blacktriangleright$, and 2) to translate the use of non-equational axioms (in particular the induction axiom of Kozen's axiomatisation) in the proof below, using purely local and equational reasoning steps.

In [38], we used a different syntax: one with another generator (also represented by a white node) interpreted as the action of regular expressions on languages. This additional generator allowed us to achieve 1), but was not sufficient to guarantee 2). Indeed, the proof of 2) is based on an incorrect claim: the rewriting procedure that is supposed to implement determinisation, as explained in the proof of [38, Lemma 4], makes unfounded assumptions on the shape of diagrams and is not guaranteed to return the desired determinisation. A counter-example is provided by the diagrammatic representation of, *e.g.*, $(aa)^*(1+a)$. The corresponding diagram should be equal to that representing $a^*$, but this cannot be proven in the equational theory of that paper—we explain this further in Example 5.21 below.

5.1. **Useful preliminaries and simplifying assumptions.** In this section, we use symmetries of the theory to make simplifying assumptions about the diagrams to consider in the completeness proof.

First, note that we need only consider equalities for completeness, since inequalities can be recovered from the semi-lattice structure of the binary operation defined by $\rightarrow\!\!\!\!\!\prec\!\!\stackrel{\rightarrow}{\phantom{a}}$ and $\stackrel{\rightarrow}{\phantom{a}}\!\!\!\succ\!\!\!\!\!-$, both semantically and syntactically as shown by the following two propositions.

**Proposition 5.2.** *For any two diagrams* $c, d$, $[\![c]\!] \subseteq [\![d]\!]$ *if and only if* $\left[\!\!\left[ \rightarrow\!\!\bullet\!\!\overbrace{\phantom{xx}}^{\;\;-c\;-}_{\;\;-d\;-}\!\!\bullet\!\!-\!\!- \right]\!\!\right] = [\![c]\!]$.

*Proof.* A routine calculation shows that $\left[\!\!\left[ \rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow \right]\!\!\right] = [\![c]\!] \cap [\![d]\!]$. So the result follows from $[\![c]\!] \cap [\![d]\!] = [\![c]\!] \Leftrightarrow [\![c]\!] \subseteq [\![d]\!]$. $\square$

**Proposition 5.3.** *For any two automata-diagrams $c, d$ we have $c \leq d$ iff* $\rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow = c$.

*Proof.* If $c \leq d$, then $c \overset{(F4)}{\leq} \rightarrow c \bullet \bigcirc \bullet \rightarrow \overset{(cpy)}{\leq} \rightarrow \bullet \overset{c}{\underset{c}{\bigcirc}} \bullet \rightarrow \leq \rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow = c$. We also

have $\rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow \overset{(F2)}{\leq} \rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow \overset{(del)}{\leq} \rightarrow \bullet \overset{c}{\bigcirc} \bullet \rightarrow$ Note that we use (cpy) and (del)
from Theorem 5.5 below (but in fact, these inequalities and the statement of the proposition, holds for *any* diagram. This can easily be proven by induction – the inductive cases are trivial, so we just have to check that the relevant inequalities holds for all generators. However, we will not need this more general fact here, as we only care about completeness for automata-diagrams).

Conversely if $\rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow = c$, then we can reason as before: $c = \rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow \overset{(F2)}{\leq}$

$\rightarrow \bullet \overset{c}{\underset{d}{\bigcirc}} \bullet \rightarrow \overset{(del)}{\leq} \rightarrow \bullet \overset{}{\underset{d}{\bigcirc}} \bullet \rightarrow = d$. $\square$

Then, we show that, without loss of generality, we can restrict our attention to diagrams of type $\blacktriangleright \rightarrow \blacktriangleright$. We proceed in two steps: 1) from all $\mathsf{Aut}_\Sigma$ diagrams to left-to-right diagrams only, and from left-to-right diagrams to those of type $\blacktriangleright \rightarrow \blacktriangleright$.

**From diagrams of $\mathsf{Aut}_\Sigma$ to left-to-right diagrams.** First, the following proposition implies that, without loss of generality, we need only consider to left-to-right diagrams (Section 4.2).

**Proposition 5.4.** *There are natural bijections between sets of string diagrams of the form*



*where $A, B, A_i, B_i$ represent lists of $\blacktriangleright$ and $\blacktriangleleft$.*

*Proof.* This proposition holds in any compact-closed category and relies on the ability to bend wires using $\overset{\frown}{\phantom{.}}$ and $\underset{\smile}{\phantom{.}}$. Explicitly, given a diagram of the first form, we can obtain one of the second form as follows:

 (5.1)

The inverse mapping is given by the same wiring with the opposite direction. That they are inverse transformations follows immediately from the defining equations of compact closed categories (A1)-(A2). The other bijection is constructed analogously. $\square$

Intuitively, Proposition 5.4 tell us that we can always bend incoming wires to the left and outgoing wires to the right before applying some equations, and recover the original orientation of the wires by bending them into their original place later.

**From left-to-right to ►→►.** As we will now show, we can further restrict our attention to diagrams ►→►. For this we prove that any left-to-right diagram $\blacktriangleright^m\to\blacktriangleright^n$ is fully characterised by $n \times m$ diagrams ►→► much like linear maps can be described by their coefficients in a given basis. Showing this amounts to proving that Lemma 4.9 extends to all left-to-right diagrams (so that the monoidal product is also a biproduct for the subcategory of left-to-right diagrams).

**Theorem 5.5** (Global distributivity). *For any automaton-diagram $d : \blacktriangleright^m\to\blacktriangleright^n$, we have*



*Proof.* According to Lemma 4.11, given $d$ as in the statement of the theorem, we can find a relation-diagram $r$ such that



$$(5.2)$$

Note first that, by Lemma 4.9, any relation-diagram satisfies (cpy) and (del) so we will use these two equations for $r$ below.
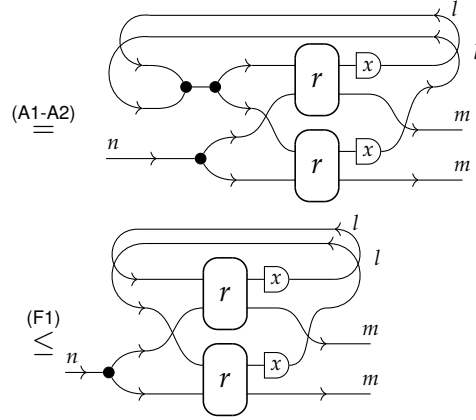
First, we prove both inequalities of (cpy).

- The first inequality requires the introduction of new black nodes, via the two axioms

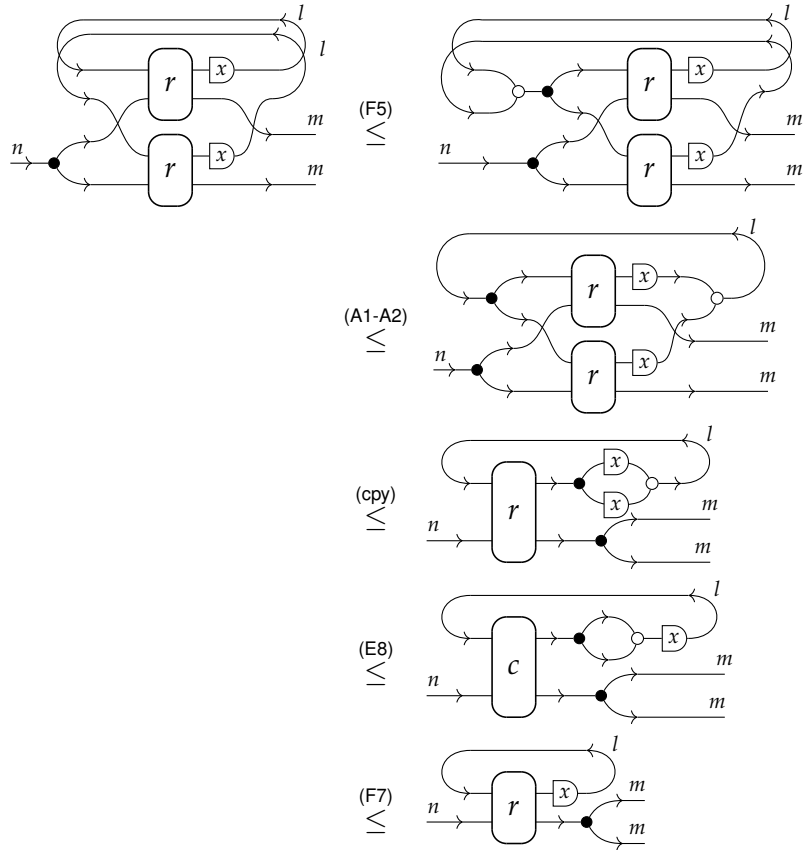- The reverse inequality requires the introduction and elimination of $\rightarrow\!\circ\!\rightarrow$, via the two axioms $\overset{(F5)}{\leq}$ and $d\rightarrow\!\bullet\!\rightarrow\overset{(F7)}{\leq}\rightarrow\!\!\!-$:

We can prove (del) in a similar way, as follows.

- The first inequality is the unary version of its (cpy) counterpart, using axioms $\longrightarrow\!\!-\ \overset{(F2)}{\leq}$
  $\rightarrow\!\!\bullet\ \bullet\!\!\rightarrow\!-$ and $\bullet\!\!\rightarrow\!\!\rightarrow\!\!\bullet\ \overset{(F4)}{\leq}\ \square$ :

$$
\overset{(F2)}{\leq}\quad\overset{(E7)}{\leq}\quad\overset{(del)}{=}\quad\overset{(A1\text{-}A2)}{=}\quad\overset{(F4)}{\leq}
$$
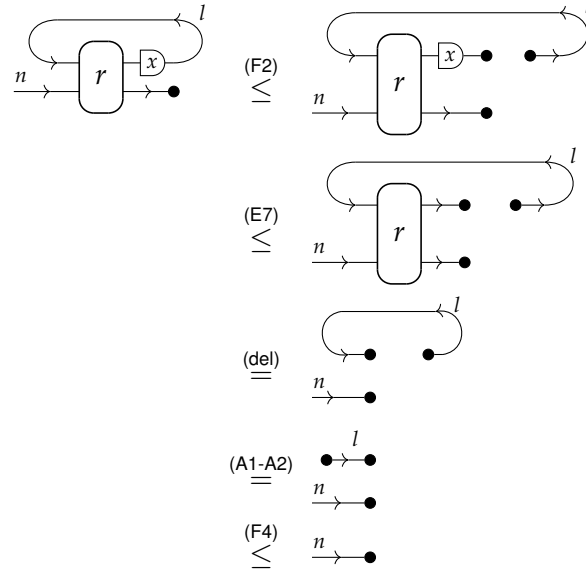
- The reverse inequality requires the introduction and elimination of $\circ\!\!\rightarrow\!-$, using axioms
  $\rightarrow\!\!\bullet\ \circ\!\!\rightarrow\!-\ \overset{(F6)}{\leq}\ \longrightarrow\!-$ and $\square\ \overset{(F8)}{\leq}\ \circ\!\!\rightarrow\!\!\bullet$ :

$$
\overset{(F6)}{\leq}\quad\overset{(A1\text{-}A2)}{\leq}\quad\overset{(del)}{\leq}\quad\overset{(E9)}{\leq}\quad\overset{(F8)}{\leq}
$$
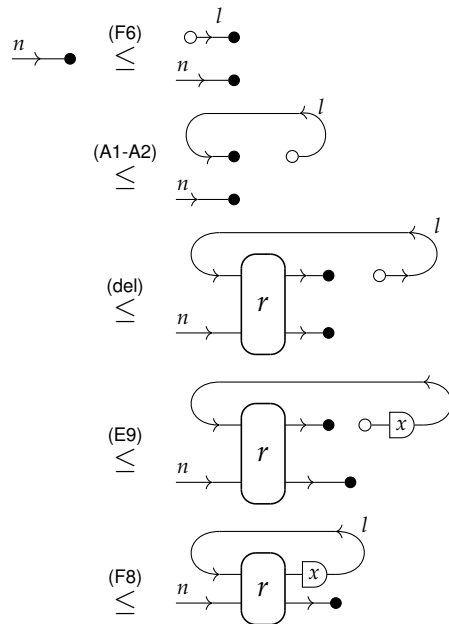
The other two equalities—(co-cpy) and (co-del)—can be proved by a symmetric argument, replacing $\rightarrow\!\!\bullet\!\!\rightarrow\!-$ with $\rightarrow\!\!\bullet\!\!\smile$, $\bullet\!\!\rightarrow\!-$ with $\rightarrow\!\!\bullet$, axioms (F9) instead of (F5), (F11) instead of (F7), (F10) instead of (F6), and (F12) instead of (F8).

□

For $d : \blacktriangleright^m \to \blacktriangleright^n$, let $d_{ij}$ be the string diagram of type $\blacktriangleright \to \blacktriangleright$ obtained as in Lemma 4.10, by discarding every input and output except the $i$-th input and $j$-th output, i.e., by composing every input with $\bullet\!\to\!-$ except the $i$-th one, and every output with $\to\!\bullet$ except the $j$-th one. Theorem 5.5 implies that left-to-right diagrams, like matrix-diagrams, are fully characterised by their $\blacktriangleright \to \blacktriangleright$ subdiagrams.

**Corollary 5.6.** *Given automata-diagrams $d, e : \blacktriangleright^m \to \blacktriangleright^n$, $d =_{KDA} e$ iff $d_{ij} =_{KDA} e_{ij}$, for all $1 \le i \le m$ and $1 \le j \le n$.*

Thus, as we claimed above, we can restrict our focus further to left-to-right $\blacktriangleright \to \blacktriangleright$ diagrams, without loss of generality. Therefore, to prove Theorem 3.3, we only need to to prove the following result.

**Theorem 5.7.** *For any two automata-diagrams $d, d' : \blacktriangleright \to \blacktriangleright$,*

$$\llbracket d \rrbracket = \llbracket d' \rrbracket \ \text{ if and only if } d =_{KDA} d'.$$

We will need to prove several preparatory results, including a diagrammatic form of determinisation, before the proof of Theorem 5.7 which can be found in Section 5.4.

5.2. **Diagrammatic subset construction.** In what follows we assume familiarity with the standard subset construction. The reader who wishes to refresh their memory can refer to [31, §6].

In diagrammatic terms, a nondeterministic transition of the automaton associated to (a representation of) a given diagram, corresponds to a subdiagram of the form $\to\!\bullet\!\overbrace{\phantom{x}}^{\boxed{a}}_{\boxed{a}}$ for some $a \in \Sigma$ in the matrix-diagram encoding its transition relation. The following example illustrates how Theorem 5.5 can already be used to determinise some simple automata-diagrams. The following section is dedicated to giving a formal procedure that extends this idea to any automaton-diagram, by formalising a diagrammatic version of the algebraic subset construction due to Kozen [29].

**Example 5.8.**



where we write $-\boxed{a^*}-$ for the left-to-right diagram  .
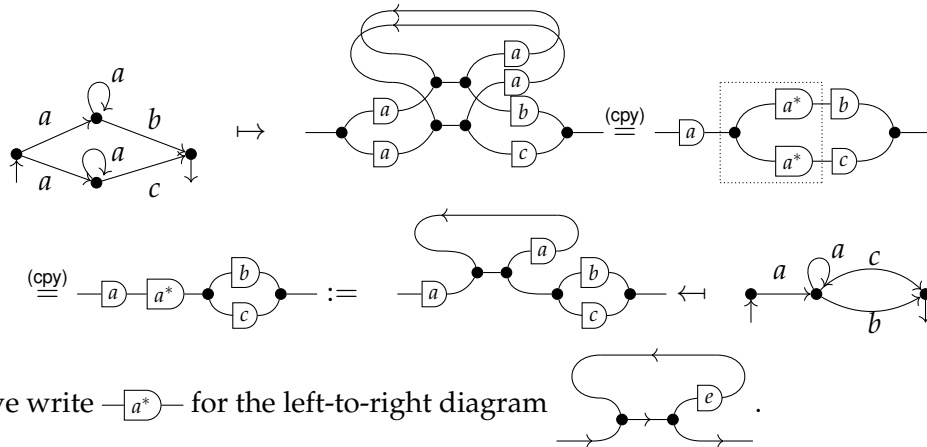
The diagrammatic counterpart of the subset construction we give below makes crucial use of the adjunctions between the different generators (Block F in Fig. 1). Before we can give the determinisation procedure for an arbitrary automaton-diagram, we need to cover essential technical preliminaries, which will allow us to greatly generalise these adjunctions.

Recall from Section 4.2 that, given a bimonoid, we can encode $n \times m$ Boolean matrices – equivalently, relations between the finite sets $\{0, \ldots, m-1\}$ and $\{0, \ldots, n-1\}$ – by a block of comultiplications and counits composed sequentially with a block of multiplications and units. The $i$-th open port on the right is connected to $j$-th one on the left iff $(i, j)$ is in the encoded relation. This time we will be working with three different bimonoids, giving three different encodings of relations: $(\rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset}, \rightarrow\!\!\bullet, \overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow, \bullet\!\!\rightarrow)$, $(\rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset}, \rightarrow\!\!\bullet, \overset{\rightarrow}{\supset}\!\!\circ\!\!\rightarrow, \circ\!\!\rightarrow)$, and $(\rightarrow\!\!\circ\!\!\overset{\rightarrow}{\subset}, \rightarrow\!\!\circ, \overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow, \bullet\!\!\rightarrow)$. We will call the corresponding matrix-diagrams $(\bullet, \bullet)$-matrices[5], $(\bullet, \circ)$-matrices and $(\circ, \bullet)$-matrices respectively.

We will now prove that the adjunctions between the white and black generators (Block F in Fig. 1) generalise to these matrix-encodings. We define two notions of transpose for diagrams: one which swaps the colours of the different generators and one which does not. These will assist us in generalising the adjunctions between the white and black generators to all our matrix encodings of relations.

**Definition 5.9.** Given a diagram $d : m \to n$, we define its *transpose* to be the diagram $d^T : n \to m$ obtained by flipping $d$ horizontally. More formally, $(\cdot)^\circ$ is defined inductively as $g^T = g$ for any generator $g$, $(\times)^T = \times$, $(c\,;d)^T = d^T\,;c^\circ$, and $(c_1 \oplus c_2)^T = c_1^T \oplus c_2^T$.

**Definition 5.10.** Given a diagram $d : m \to n$, we define its *colour-transpose* to be the diagram $d^\circ : n \to m$ obtained from $d$ by swapping all black and white nodes, and flipping the resulting diagram horizontally. More formally, $(\cdot)^\circ$ is defined inductively as follows:

$$(\rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset})^\circ = \overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow \quad (\rightarrow\!\!\bullet)^\circ = \circ\!\!\rightarrow \quad (\overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow)^\circ = \rightarrow\!\!\circ\!\!\overset{\rightarrow}{\subset} \quad (\bullet\!\!\rightarrow)^\circ = \rightarrow\!\!\circ$$

$$(\rightarrow\!\!\circ\!\!\overset{\rightarrow}{\subset})^\circ = \overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow \quad (\rightarrow\!\!\circ)^\circ = \bullet\!\!\rightarrow \quad (\overset{\rightarrow}{\supset}\!\!\circ\!\!\rightarrow)^\circ = \rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset} \quad (\circ\!\!\rightarrow)^\circ = \rightarrow\!\!\bullet$$

$$(\times)^\circ = \times \quad (c\,;d)^\circ = d^\circ\,;c^\circ \quad (c_1 \oplus c_2)^\circ = c_1^\circ \oplus c_2^\circ$$

Note that a permutation $\sigma$ is mapped to its inverse so that $\sigma^T = \sigma^\circ = \sigma^{-1}$. Another immediate consequence of these definitions is that these mappings are involutive: $(d^\circ)^\circ = d$ and $(d^T)^T = d$.

For the next lemmas, we use the following notation. Given some relation, we write $-\!\!\blacktriangleright\!\!-$ for the corresponding $(\bullet, \bullet)$-matrix. We can assume it factors as $-\!\!\blacktriangleright\!\!- = b\,;c$, where $b$ is a diagram formed only of $\rightarrow\!\!\bullet\!\!\overset{\rightarrow}{\subset}, \rightarrow\!\!\bullet$ and $c$ is a diagram formed only of $\overset{\rightarrow}{\supset}\!\!\bullet\!\!\rightarrow, \bullet\!\!\rightarrow$. Then, we define $-\!\!\text{◗}\!\!- := c^\circ\,;b^T$ and $-\!\!\text{◖}\!\!- := c^T\,;b^\circ$.

It is helpful to adopt a semantic point of view in order to develop intuition about these different diagrammatic encodings of relations. If $-\!\!\blacktriangleright\!\!-$ encodes the relation $R$, we have

$$[\![-\!\!\blacktriangleright\!\!-]\!] = \{(\mathbf{L}, \mathbf{K}) \mid L_i \subseteq K_j \text{ if } (i, j) \in R\}$$

$$[\![-\!\!\triangleright\!\!-]\!] = \left\{(\mathbf{L}, \mathbf{K}) \mid L_i \subseteq \bigcap_{(i,j)\in R} K_j\right\} \qquad [\![-\!\!\triangleright\!\!-]\!] = \left\{(\mathbf{L}, \mathbf{K}) \mid \bigcup_{(i,j)\in R} L_i \subseteq K_j\right\}$$

Furthermore, the matrix $-\!\!\blacktriangleright\!\!-$ can be seen as the embedding of a monotone map $\mathcal{L}_\Sigma^m \to \mathcal{L}_\Sigma^n$ (see Remark 2.3) in monotone relations. In fact, the corresponding map is not just monotone but a lattice homomorphism, *i.e.* a map that preserves both meets/intersections and joins/unions. As a result, it admits both left $(-\!\!\triangleright\!\!-)$ and right $(-\!\!\triangleright\!\!-)$ adjoints.

From an equational perspective, as we will now prove, the F axioms of KDA are enough to derive $-\!\!\triangleright\!\!- \dashv -\!\!\blacktriangleright\!\!- \dashv -\!\!\triangleright\!\!-$.

---

[5]These are the *relation-diagrams* of Section 4.3.

**Lemma 5.11.** *If $b : m \to n$ is a diagram made entirely from $\rightarrow\!\!\curlywedge$ and $\rightarrow\!\!\bullet$, we have*

$$(i) \quad \xrightarrow{\;n\;}\!\!- \;\leq\; \xrightarrow{\;n\;}\!\!\boxed{b^\circ}\!\!-\!\!\boxed{b}\!\!\xrightarrow{\;n\;} \qquad and \qquad (ii) \quad \xrightarrow{\;m\;}\!\!\boxed{b}\!\!-\!\!\boxed{b^\circ}\!\!\xrightarrow{\;m\;} \;\leq\; \xrightarrow{\;m\;}\!\!-$$

*Proof.* Consider inequality *(i)*. This can be proven by a straightforward induction on the structure of $b$. We can take care of the base cases with axioms (F5) and (F8). There are three inductive cases to consider, which can be dealt with using these two axioms again:

- $b = {}_{m-1}\!\overset{\rightarrow}{\underset{\rightarrow}{\curlywedge}}\boxed{b'}\!\!\xrightarrow{\,n\,}$. Then $b^\circ \,;\, b = \xrightarrow{\,n\,}\boxed{(b')^\circ}\!\!\overset{\rightarrow}{\underset{m-1}{\curlyvee\!\curlywedge}}\boxed{b'}\!\!\xrightarrow{\,n\,} \overset{(F5)}{\geq} (b')^\circ \,;\, b'.$

- $b = {}_{m-1}\!\overset{\rightarrow\bullet}{\underset{\rightarrow}{}}\boxed{b'}\!\!\xrightarrow{\,n\,}$. Then $b^\circ \,;\, b = \xrightarrow{\,n\,}\boxed{(b')^\circ}\!\!\overset{\circ\rightarrow\bullet}{\underset{m-1}{}}\boxed{b'}\!\!\xrightarrow{\,n\,} \overset{(F8)}{\geq} (b')^\circ \,;\, b'$

- $b = \sigma \,;\, b'$ for some permutation $\sigma$. Then $b^\circ \,;\, b = (b')\circ \,;\, \sigma^{-1} \,;\, \sigma \,;\, b' = (b')^\circ \,;\, b'.$

In all three cases we can conclude that $\xrightarrow{\,n\,}\!\!- \leq \xrightarrow{\,n\,}\boxed{b^\circ}\!\!-\!\!\boxed{b}\!\!\xrightarrow{\,n\,}$ using the induction hypothesis.

Inequality *(ii)* can also be proven by a similar induction. We can take care of the base cases with axioms (F6) and (F7). As before, there are three inductive cases to consider, which can be dealt with using these two axioms again:

- $b = {}_{m-1}\!\overset{\rightarrow\bullet}{\underset{\rightarrow}{\curlywedge}}\boxed{b'}\!\!\xrightarrow{\,n\,}$. Then $b \,;\, b^\circ = {}_{m-1}\!\overset{\rightarrow\bullet}{\curlywedge}\boxed{b'}\!\!\xrightarrow{\,n\,}\boxed{(b')^\circ}\!\!\underset{m-1}{\curlyvee} \overset{(I.H.)}{\leq} \overset{\rightarrow\bullet\curlyvee\rightarrow}{\underset{m-1}{}} \overset{(F7)}{\geq} \xrightarrow{\,m\,}\!\!-.$

- $b = {}_{m-1}\!\overset{\rightarrow\bullet}{\underset{\rightarrow}{}}\boxed{b'}\!\!\xrightarrow{\,n\,}$. Then $b \,;\, b^\circ = {}_{m-1}\!\overset{\rightarrow\bullet}{}\boxed{b'}\!\!\xrightarrow{\,n\,}\boxed{(b')^\circ}\!\!\overset{\circ\rightarrow}{\underset{m-1}{}} \overset{(I.H.)}{\leq} \overset{\rightarrow\bullet\,\circ\rightarrow}{\underset{m-1}{}} \overset{(F6)}{\leq} \xrightarrow{\,m\,}\!\!-$

- $b = \sigma \,;\, b'$ for some permutation $\sigma$. Then $b^\circ \,;\, b = \sigma \,;\, b \,;\, b^\circ \,;\, \sigma^{-1} \overset{(I.H.)}{\leq} \sigma^{-1} \,;\, \sigma = \xrightarrow{\,m\,}\!\!-.$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

**Lemma 5.12.** *If $w : m \to n$, is a diagram made entirely from $\rightarrow\!\!\curlywedge$ and $\rightarrow\!\!\circ$, we have*

$$(i) \quad \xrightarrow{\;n\;}\!\!- \;\leq\; \xrightarrow{\;n\;}\!\!\boxed{w^\circ}\!\!-\!\!\boxed{w}\!\!\xrightarrow{\;n\;} \qquad and \qquad (ii) \quad \xrightarrow{\;m\;}\!\!\boxed{w}\!\!-\!\!\boxed{w^\circ}\!\!\xrightarrow{\;m\;} \;\leq\; \xrightarrow{\;m\;}\!\!-$$

*Proof.* The proof is entirely analogous to that of the previous lemma. Inequality *(i)* can be proven in the same way, replacing all uses of (F5) by (F9) and (F8) by (F12). Similarly, we can prove inequality *(ii)* by replacing all uses of (F6) by (F10) and (F7) by (F11). $\quad\square$

**Lemma 5.13.** *If $b : m \to n$ is a diagram made entirely from $\rightarrow\!\!\curlywedge$ and $\rightarrow\!\!\bullet$, we have*

$$(i) \quad \xrightarrow{\;n\;}\!\!- \;\leq\; \xrightarrow{\;m\;}\!\!\boxed{b}\!\!-\!\!\boxed{b^T}\!\!\xrightarrow{\;m\;} \qquad and \qquad (ii) \quad \xrightarrow{\;n\;}\!\!\boxed{b^T}\!\!-\!\!\boxed{b}\!\!\xrightarrow{\;n\;} \;\leq\; \xrightarrow{\;m\;}\!\!-$$

*Proof.* The proof is once again entirely analogous to that of the Lemma 5.12, with the converse inequalities (as $\rightarrow\!\!\curlywedge$ is left adjoint to $\overset{\rightarrow}{\underset{\rightarrow}{\bullet\!\!-}}$, whereas $\rightarrow\!\!\curlywedge$ is right adjoint to $\overset{\rightarrow}{\underset{}{\circ\!\!-}}$). We can prove inequality *(i)* as inequality *(ii)* in Lemma 5.12 by replacing all uses of (F6) by (F2) and (F7) by (F3). Inequality *(ii)* can be proven in the same way as inequality *(i)* in Lemma 5.12, replacing all uses of (F5) by (F1) and (F8) by (F4). $\quad\square$

Recall that if $-\!\!\blacksquare\!\!- = b \,;\, c$, $-\!\!\blacksquare\!\!- := c^\circ \,;\, b^T$ and $-\!\!\blacksquare\!\!- := c^T \,;\, b^\circ$.

**Lemma 5.14.** *(i)* $\xrightarrow{\;m\;}\!\!- \;\leq\; -\!\!\blacksquare\!\!\blacksquare\!\!-$ *and (ii)* $-\!\!\blacksquare\!\!\blacksquare\!\!- \;\leq\; \xrightarrow{\;n\;}\!\!-$

*Proof.* Let $w = c^\circ$ so that $w$ is made entirely from $\rightarrow\!\!\curlywedge, \rightarrow\!\!\circ$ (and we can apply Lemma 5.12 to it) and notice that $w^\circ = (c^\circ)^\circ = c$.

For *(i)*, we have $\xrightarrow{m}\!\!\!-\ \leq\ b\,;\,b^T\ \leq\ b\,;\,w^\circ\,;\,w\,;\,b^T\,:=\,b\,;\,c\,;\,c^\circ\,;\,b^T\,=:\ \ \blacksquare\!\!-\!\!\blacksquare\ $ where the first inequality comes from Lemma 5.13 *(i)* and the second from Lemma 5.12 *(i)*.

For *(ii)*, we have $-\!\blacksquare\!-\!\blacksquare\!-\ :=\ c^\circ\,;\,b^T\,;\,b\,;\,c\ \leq\ c^\circ\,;\,c\,=\,w\,;\,w^\circ\ \leq\ \xrightarrow{n}\!\!\!-\ $ where the first inequality comes from Lemma 5.13 *(ii)* and the second one from Lemma 5.12 *(ii)*. $\qquad\square$

**Lemma 5.15.** *(i)* $\xrightarrow{n}\!\!\!-\ \leq\ -\!\blacksquare\!-\!\blacksquare\!-$ *and (ii)* $-\!\blacksquare\!-\!\blacksquare\!-\ \leq\ \xrightarrow{m}\!\!\!-$

*Proof.* Let $b_1 = c^T$ and notice that $b_1^T = (c^T)^T = c$.

For *(i)*, we have $\xrightarrow{n}\!\!\!-\ \leq\ b_1\,;\,b_1^T\,:=\,c^T\,;\,c\ \leq\ c^T\,;\,b^\circ\,;\,b\,;\,c\ =:\ \ -\!\blacksquare\!-\!\blacksquare\!-\ $ where the first inequality comes from Lemma 5.13 *(i)* and the second from Lemma 5.11 *(i)*.

For *(ii)*, we have $-\!\blacksquare\!-\!\blacksquare\!-\ :=\ b\,;\,c\,;\,c^T\,;\,b^\circ\ =:\ b\,;\,b_1^T\,;\,b_1\,;\,b^\circ\ \leq\ b\,;\,b^\circ\ \leq\ \xrightarrow{n}\!\!\!-\ $ where the first inequality comes from Lemma 5.13 *(ii)* and the second one from Lemma 5.11 *(ii)*. $\qquad\square$

We will also need the following generalised distributivity (in)equalities.

**Lemma 5.16.** *(i)* $\overset{\rightarrow}{\Rightarrow}\!\!\bullet\!\!-\!\blacksquare\!-\ \leq\ \overset{\rightarrow}{\Rightarrow}\!\!\bullet\!\!\rightarrow$ *and (ii)* $\overset{\rightarrow}{\Rightarrow}\!\!\bullet\!\!-\!\blacksquare\!-\ =\ \overset{\rightarrow}{\Rightarrow}\!\!\bullet\!\!\rightarrow$

*Proof.* Inequality *(i)* is a standard fact in a Cartesian bicategory and can be proven by structural induction on diagrams made exclusively from the generators $\rightarrow\!\!\bullet\!\!\overset{\frown}{\subset}\,,\rightarrow\!\!\bullet\,,\overset{\rightarrow}{\Rightarrow}\!\!\rightarrow\,,\circ\!\!\rightarrow$. The base cases are a consequence of axioms (B7)-(B9) and (E1)-(E2), while the two inductive cases (for vertical and horizontal composition) are immediate.

Equality *(ii)* is a standard consequence of the bimonoid axioms and can be proven by structural induction on diagrams made exclusively from the generators $\rightarrow\!\!\bullet\!\!\overset{\frown}{\subset}\,,\rightarrow\!\!\bullet\,,\overset{\rightarrow}{\Rightarrow}\!\!\rightarrow\,,\circ\!\!\rightarrow$. the base cases are a consequence of axioms (D4)-(D6), while the two inductive cases (for vertical and horizontal composition) are immediate. $\qquad\square$

Following Kozen's proof of completeness [29] we can model the subset construction algebraically, now with diagrams. We can construct a $(\bullet,\bullet)$-matrix $p_s : \blacktriangleright^s \to \blacktriangleright^{2^s}$ encoding the membership relation of elements of $\{0,\ldots,s-1\}$ to subsets of $\{0,\ldots,s-1\}$. In diagrammatic terms, the $j$-th port on the left is connected to $i$-th one on the right iff $j \in i \subseteq \{0,\ldots,s-1\}$. For example, $p_2$ is the following diagram
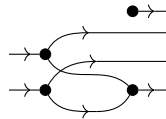


where the ports on the left correspond to 0 and 1, and those on the right correspond to the subsets $\varnothing$, $\{0\}$, $\{1\}$, and $\{0,1\}$, from top to bottom.

From now on $-\!\blacksquare\!-$ will refer to $p_S$, $-\!\blacksquare\!-$ to the corresponding $(\bullet,\circ)$-matrix, and $-\!\blacksquare\!-$ to the corresponding $(\circ,\bullet)$-matrix, constructed as explained above. We will also need the diagrams for the transposes of these matrices: they have type $\blacktriangleright^{2^s} \to \blacktriangleright^s$ and we will use the notations $-\!\blacksquare\!-$, $-\!\square\!-$, and $-\!\square\!-$, often omitting the explicit label $s$.

We need a similar matrix-diagram for the subset construction applied to the initial state. Let $e_i : \blacktriangleright^s \to \blacktriangleright^{2^s}$ be the matrix-diagram whose $i$-th port on the left is connected to the port on the right encoding the singleton subset $\{i\}$ (which we can choose to be the $i+1$-th port on the right by fixing an ordering of the right ports that starts with $\varnothing, \{1\}, \{2\}, \ldots$). We

will use this to encode the initial state $\hat{e}$ of the determinisation of a given diagram with representation $(d, e, f)$: by construction

$$\hat{e} = e \,;\, e_i \tag{5.3}$$

The next two lemmas connect the diagrammatic representation of a given automaton to that of its determinisation. They are simply a reformulation of Kozen's construction in [29]. For both statements we assume that $d : \blacktriangleright^s \to \blacktriangleright^s$, $f : \blacktriangleright^s \to \blacktriangleright$, $e : \blacktriangleright \to \blacktriangleright^s$ are matrix-diagrams encoding the transition relation, final and initial states of a given automaton, and $\hat{d} : \blacktriangleright^{2^s} \to \blacktriangleright^{2^s}$, $\hat{f} : \blacktriangleright^{2^s} \to \blacktriangleright$, $\hat{e} : \blacktriangleright \to \blacktriangleright^{2^s}$ are matrix-diagrams encoding the transition relation, final and initial state of its determinisation, respectively.

**Lemma 5.17.**



*and*



*Proof.* By construction of  . The claim for the corresponding matrix can be found in [29, Lemma 17]. The same diagrammatic fact holds because of the completeness of our theory for $(\bullet, \bullet)$-matrix-diagrams (Theorem 4.8). $\qquad\square$

**Lemma 5.18.** *(i)*  $\leq$  *and (ii)*  $\leq$  *(notice the difference between*  *and*  *).*

*Proof.* For the first inequality, $e \,;\,$  contains the following sub-diagram, where $i$ is the only right port of $e$ connected to its single left port (the wire encoding the initial state in this representation), and $j$ is another arbitrary wire (and not a stack of $j$ wires here):
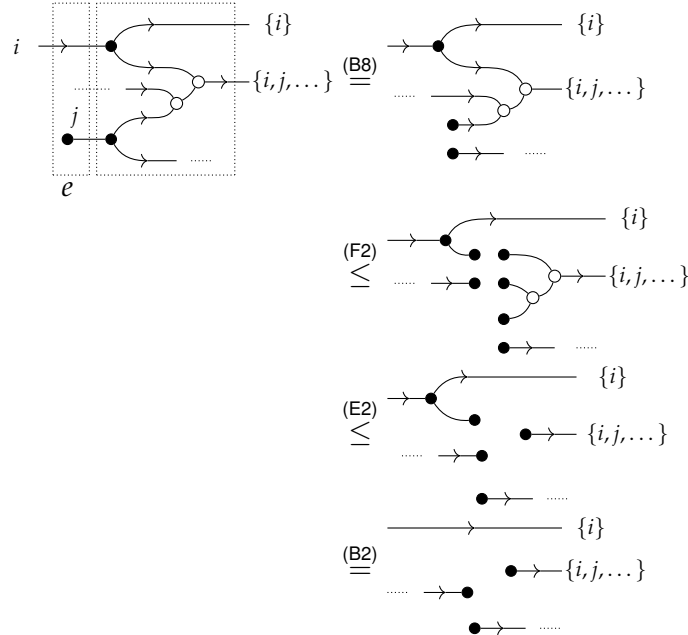


We can perform this sequence of steps for any $j \neq i$: as a result, we are left with a $(\bullet, \bullet)$ matrix-diagram $1 \to 2^s$ whose only left port is connected to the port corresponding to $\{i\}$ on the right. This is precisely $\hat{e}$, as expected.

For the second inequality, we can reason similarly, starting form the following sub-diagram of $e$ ; $-\!\blacksquare\!-$, with $i$ the wire representing the initial state as before, and $j$ some other arbitrary wire:
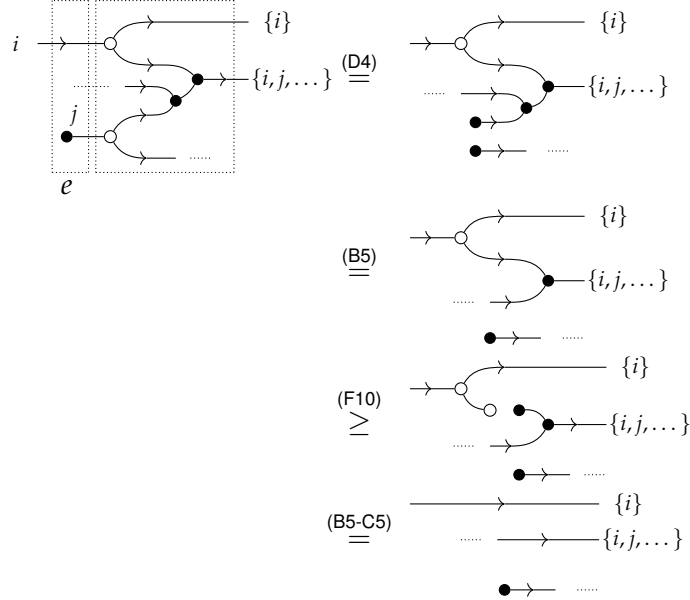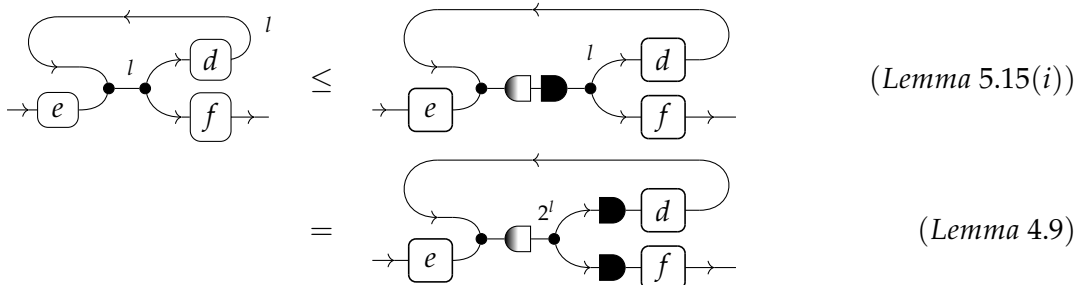


We can perform the same sequence of steps for any $j \neq i$. The number of $\overrightarrow{\smash{\;}}\!\bullet\!\succ$ connected to the wire labelled $\{i, j, \dots\}$ above decreases at every step. Thus, as before, in the end we are left with a $(\bullet, \bullet)$ matrix-diagram $\blacktriangleright\!\rightarrow\!\blacktriangleright^{2^s}$ whose only left port is connected to the port encoding the singleton $\{i\}$ on the right. This is precisely $\hat{e}$ again. $\qquad \square$

5.3. **Determinisation.** We are now able to devise a determinisation procedure for representation of automata-diagrams. One of the payoffs of our approach is that the proof of the following theorem can be carried out purely equationally: the adjunctions we have constructed in the previous subsection make it possible to replace Kozen's use of bisimulation laws in his completeness proof [29] by local diagrammatic rewriting steps.

**Theorem 5.19.** *Every automaton-diagram is equal to its determinisation.*

*Proof.* We show both inequalities separately.

- Let us start by proving that a given automata-diagram with representation $(e, d, f)$ is included in its determinisation $(\hat{e}, \hat{d}, \hat{f})$.

$\leq$  (*Lemma* 5.16)(*i*)

$=$  (*Compactness*)

$=$  (*Lemma* 5.17(*i*))

$\leq$  (*Lemma* 5.15(*ii*))

$\leq$  (*Lemma* 5.18(*i*))

$=$  (*Lemma* 5.17(*ii*))

- For the reverse inclusion, we have

 $=$  (*Lemma* 5.17(*ii*))

$\leq$  (*Lemma* 5.18(*ii*))

$\leq$  (*Lemma* 5.14(*i*))

$=$  (*Lemma* 5.17(*i*))

$$= \qquad (Compactness)$$

$$= \qquad (Lemma\ 5.16(ii))$$

$$= \qquad (Lemma\ 4.9)$$

$$= \qquad (Lemma\ 5.14(ii))$$

$\square$
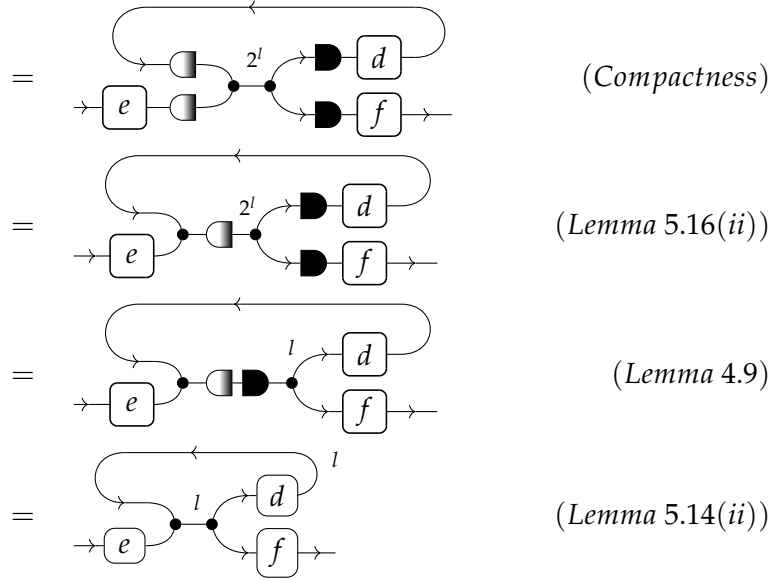
Combining the theorem above with the existence of representations (Theorem 4.7) yields the result we are after.

**Corollary 5.20** (Determinisation). *Any automaton-diagram ▶→▶ has a deterministic represen-tation.*

5.4. **Minimisation and completeness.** As explained above, our proof of completeness is a diagrammatic reformulation of Brzozowski's algorithm, which proceeds in four steps: determinise, reverse, determinise, reverse. We already know how to determinise a given diagram. The other three steps are simply a matter of changing our perspective on diagrams, looking at them from right to left, and noticing that all the equations that we needed to determinise them, can be performed in reverse.

We say that a matrix-diagram is *co-deterministic* if the converse of its associated transition relation is deterministic.

*Proof of Theorem 5.7 (Completeness).* We have a procedure to show that, if $[\![d]\!] = [\![d']\!]$, then there exists a string diagram $c$ in normal form such that $d = c = d'$. This normal form is the diagrammatic counterpart of the *minimal* automaton associated to $d$ and $d'$. In our setting, it is the deterministic representation of $d$ and $d'$ with the smallest number of states. This is unique because we can obtain from it the corresponding minimal automaton, which is well-known to be unique. First, given any string diagram we can obtain a representation for it by Proposition 4.7. Then we obtain a minimal representation by splitting Brzozowski's algorithm in two steps.

**1. Reverse; determinise; reverse:** A close look at the determinisation procedure shows that, at each step, the required equations all hold in reverse, read from right to left instead of left to right. For example, we can replace every instance of (cpy) with (co-cpy). We can thus define, in a completely analogous manner, a co-determinisation procedure which takes care of the first three steps of Brzozowski's algorithm, and obtain a co-deterministic representation for the given diagram.

**2. Determinise:** By applying Corollary 5.20, we can obtain a deterministic representation from the co-deterministic representation of the previous step. The result is the desired minimal deterministic representation and normal form.

<div style="text-align: right;">□</div>

**Example 5.21.** This example treats the diagrammatic equivalent of the regex $(aa)^*(1+a)$ which denotes the same language as $a^*$. This is a simple example of an equivalence that cannot be proven in Kleene algebra without the induction axiom (or some equivalent infinitary axiom scheme encoding induction). We prove the first inclusion below.
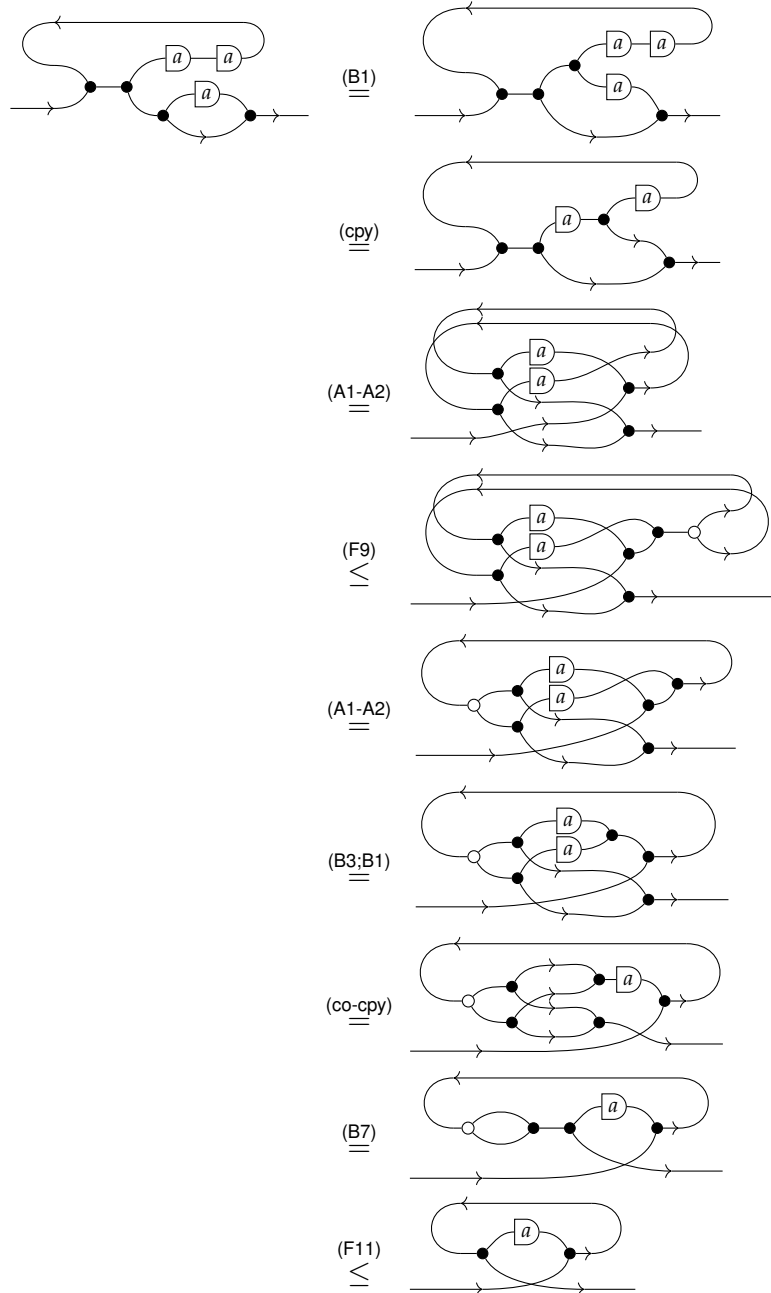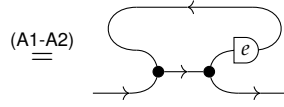
The reverse inclusion can be proven similarly, by replacing (F9) with (F1) and (F11) with (F3).

This is an example of an equality that could not be proven in the equational theory of the conference paper [38]: the determinisation procedure proposed in the proof of [38, Lemma 4] would fail to identify the two equivalent states (represented by the two loops in the representation obtained on the third line of the derivation above) and get stuck.

In Section 4.2, we explored the correspondence between ▶→▶ diagrams and regular expressions. In the light of our completeness result, we can revisit this correspondence and extend it to arbitrary left-to-right diagrams $\blacktriangleright^m \to \blacktriangleright^n$. More precisely, we can now prove a *Kleene theorem* for left-to-right diagrams. We can extend the notion of matrix-diagram (Definition 4.4) to that of *regex matrix-diagram*, which is a left to right diagram that factors as a block of $\to\bullet\!\!\overset{\curvearrowright}{\phantom{.}}$, $\to\bullet$, followed by a block of regex-diagrams and finally, a block of $\overset{\to}{\phantom{.}}\bullet\!\!\vdash$, $\bullet\!\!\vdash$. This is the diagrammatic counterpart of a matrix with regex coefficients. The completeness of KDA implies that every left-to-right diagram can be put in this form.

**Corollary 5.22** (Kleene theorem for Aut$_\Sigma$). *Any left-to-right diagram is equal to a regex matrix-diagram.*

*Proof.* Let $d : \blacktriangleright^m \to \blacktriangleright^n$ be a left-to-right diagram. By Corollary 5.6, $d$ is fully characterised by its coefficients $d_{ij}$ obtained by discarding all but one of the left ports and all but one of the right ports. According to Proposition 4.6, we can find a representation for each $d_{ij}$ and therefore a regular language $L_{ij}$ recognised by the associated automata. By the standard version of Kleene theorem, we can pick a regex $e_{ij}$ that describes $L_{ij}$. Then, by soundness $[\![d_{ij}]\!] = [\![\langle e_{ij}\rangle]\!]$ and by completeness $d_{ij} = \langle e_{ij}\rangle$. This shows that $d$ is equal to a diagram that factors as a matrix of regex-diagrams, as we wanted to prove. □

As a result, any given $\blacktriangleright^m \to \blacktriangleright^n$ diagram is fully characterised by an $n \times m$ array of regular languages. Finally, by Theorem 5.4 any given diagram (not necessarily left-to-right) with $m$ inputs and $n$ outputs is fully characterised by an array $n \times m$ array of regular languages and where each of the inputs and outputs is located (on the left or on the right interface).

## 6. DISCUSSION

In this paper, we have given a fully diagrammatic treatment of finite-state automata, with a finite equational theory that axiomatises them up to language equivalence. We have seen that this allows us to decompose the regular operations of Kleene algebra, like the star, into more primitive components, resulting in greater modularity. In this section, we compare our contributions with related work, and outline directions for future research.

Traditionally, computer scientists have used *syntax diagrams* (also called *railroad diagrams*) to visualise regular expressions and context-free grammars [47]. These diagrams resemble ours very closely but have remained mostly informal More recently, Hinze has treated the single input-output case rigorously as a pedagogical tool to teach the correspondence between finite-state automata and regular expressions [21]. He did not, however, study their equational properties.

Bloom and Ésik's *iteration theories* provide a general categorical setting in which to study the equational properties of iteration for a broad range of structures that appear in programming languages semantics [6]. They are cartesian categories equipped with a parameterised fixed-point operation closely related to the feedback notion we have used to represent the Kleene star. However, the monoidal category of interest in this paper is *compact-closed*, a property that is incompatible with the existence of categorical products (any compact-closed category for which the monoidal product is also the categorical product is trivial [34]). Nevertheless, the subcategory of left-to-right diagrams (Section 4.2) is a (matrix) iteration theory [7], a structure that Bloom and Ésik have used to give an (infinitary) axiomatisation of regular languages [5].

Similarly, Stefanescu's work on *network algebra* provides a unified algebraic treatment of various types of networks, including finite-state automata [45]. In general, network algebras are traced monoidal categories where the product is not necessarily cartesian, and therefore more general than iteration theories. In both settings however, the trace is a global operation, that cannot be decomposed further into simpler components. In our work, on the other hand, the trace can be defined from the compact-closed structure, as was depicted in (1.3).

Note that the compact closed category in this paper can be recovered from the traced monoidal category of left-to-right diagrams, via the *Int construction* [25]. Therefore, as far as mathematical expressiveness is concerned, the two approaches are equivalent. However, from a methodological point of view, taking the compact closed structure as primitive allows for improved compositionality, as example (1.2) in the introduction illustrates. Furthermore, the compact closed structure can be finitely presented relative to the theory of symmetric monoidal categories, whereas the trace operation cannot. This matters greatly in this paper, where finding a finite axiomatisation is our main concern.

In all the formalisms we have mentioned, the difficulty typically lies in capturing the behaviour of iteration—whether as the star in Kleene algebra [29, 5], or a trace operator [6] in iteration theory and network algebra [45]. The axioms should be coercive enough to force it to be *the least fixed-point* of the language map $L \mapsto \{\epsilon\} \cup LK$. In Kozen's axiomatisation of Kleene algebra [29] for example, this is through (a) the axiom $1 + ee^* \leq e^*$ (star is a fixpoint) and (b) the Horn clause $f + ex \leq x \Rightarrow e^*f \leq x$ (star is the least fixpoint). In our work, (a) is a consequence of the unfolding of the star into a feedback loop and can be derived from the axioms involving only the automata generators (black nodes); (b) on the other hand does require the existence of their adjoints (white nodes).

Pratt's *action algebras* achieve a similar technical goal: the algebraic theory of action algebra is a finitely-based conservative extension of Kleene algebra [39]. An action algebra is a Kleene algebra and a residuated lattice: it has two additional operations of implication that are adjoint to left/right multiplication. As in our setting, the induction axiom of action algebras can be derived from a finite number of purely equational axioms. However, contrary to the theory of Kleene algebra, equality for action algebras is undecidable – it remains to be seen whether that is also the case for the whole language of this paper, and we leave investigation of its decidability for future work.

In the conference paper [38] on which this work is based, we presented a finite equational theory for $\mathrm{Aut}_\Sigma$ only (without the white nodes that we use in this work). Unfortunately, this theory turns out to not be complete. Example 5.21 provides a counter-example to the claim of completeness of [38]. There are several ways to fix the issue. The first would be to recast the existing infinitary axiomatisations of the matricial iteration theories of regular

languages [5] into our diagrammatic framework. This would simply involve restating the two axiom schemes characterising the behaviour of the Kleene star as equations about feedback loops, leading to an infinitary axiomatisation. While this is certainly feasible, we wanted to achieve a *finitary* presentation, which has led us to the approach developed in the present paper. By extending the syntax, we have been able to exploit additional structure over the set of regular languages to obtain a finite theory.

There is an intriguing parallel between our case study and the positive fragment of relation algebra (also known as allegories [18]). Indeed, allegories, like Kleene algebra, do not admit a finite axiomatisation [18]. However, this result holds for standard algebraic theories. It has been shown recently that a structure equivalent to allegories can be given a finite axiomatisation when formulated in terms of string diagrams in monoidal categories [10]. It seems like the greater generality of the monoidal setting—algebraic theories correspond precisely to the particular case of cartesian monoidal categories [12]—allows for simpler axiomatisations in some specific cases. In the future we would like to understand whether this phenomenon, of which now we have two instances, can be understood in a general context.

Lastly, extensions of Kleene Algebra, such as Concurrent Kleene Algebra (CKA) [22, 26] and NetKAT [1], are increasingly relevant in current research. Enhancing our theory $=_{KDA}$ to encompass these extensions seems a promising research direction, for two main reasons. First, the two-dimensional nature of string diagrams has been proven particularly suitable to reason about concurrency (see e.g. [8, 44]), and more generally about resource exchange between processes (see e.g. [11, 15, 24, 4, 9]). Second, when trying to transfer the good meta-theoretical properties of Kleene Algebra (like completeness and decidability) to extensions such as CKA and NetKAT, the cleanest way to proceed is usually in a modular fashion. The interaction between the new operators of the extension and the Kleene star usually represents the greatest challenge to this methodology. Now, in $=_{KDA}$, the Kleene star is decomposable into simpler components (see (1.3)). We believe this is a particularly favourable starting point to modularise a meta-theoretic study of CKA and NetKAT with string diagrams, taking advantage of the results we presented in this paper for finite-state automata.

In this work, we have left open the question of completeness for the whole language, including the white generators of (2.2). In an upcoming paper [19] we give a partial answer for a restricted fragment: the same theory without the letters (*i.e.* over an empty alphabet), and with the addition of axioms turning the white generators into a Frobenius monoid, is complete for monotone relations between Boolean algebras. Of course, we expect the case of nonempty alphabets to be more complicated. We leave this for further work, starting with the simpler case of a single-letter alphabet.

## References

[1] Anderson, C.J., Foster, N., Guha, A., Jeannin, J.B., Kozen, D., Schlesinger, C., Walker, D.: Netkat: semantic foundations for networks. ACM SIGPLAN Notices **49**(1), 113–126 (2014)

[2] Arden, D.N.: Delayed-logic and finite-state machines. In: 2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961). pp. 133–151. IEEE (1961)

[3] Baez, J., Coya, B., Rebro, F.: Props in network theory. Preprint arXiv:1707.08321 (2017)

[4] Baez, J.C., Fong, B.: A compositional framework for passive linear networks. Theory & Applications of Categories **33** (2018)

[5] Bloom, S.L., Ésik, Z.: Equational axioms for regular sets. Mathematical structures in computer science **3**(1), 1–24 (1993)

[6] Bloom, S.L., Ésik, Z.: Iteration theories. Springer (1993)

[7] Bloom, S.L., Ésik, Z.: Matrix and matricial iteration theories. Journal of Computer and System Sciences **46**(3), 381–439 (1993)

[8] Bonchi, F., Holland, J., Piedeleu, R., Sobociński, P., Zanasi, F.: Diagrammatic algebra: from linear to concurrent systems. In: Proceedings of the 46th Annual ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) (2019)

[9] Bonchi, F., Piedeleu, R., Sobociński, P., Zanasi, F.: Graphical affine algebra. In: Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (2019)

[10] Bonchi, F., Seeber, J., Sobocinski, P.: Graphical conjunctive queries. In: 27th Annual EACSL Conference Computer Science Logic, (CSL). vol. 119 (2018)

[11] Bonchi, F., Sobociński, P., Zanasi, F.: The calculus of signal flow diagrams I: linear relations on streams. Information and Computation **252**, 2–29 (2017)

[12] Bonchi, F., Sobociński, P., Zanasi, F.: Deconstructing Lawvere with distributive laws. Journal of logical and algebraic methods in programming **95**, 128–146 (2018)

[13] Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. Mathematical theory of Automata **12**(6), 529–561 (1962)

[14] Carboni, A., Walters, R.: Cartesian bicategories I. Journal of pure and applied algebra **49**(1-2), 11–32 (1987)

[15] Coecke, B., Kissinger, A.: Picturing Quantum Processes - A first course in Quantum Theory and Diagrammatic Reasoning. Cambridge University Press (2017)

[16] Conway, J.H.: Regular algebra and finite machines. Courier Corporation (2012)

[17] Fong, B., Spivak, D.I.: An invitation to applied category theory: seven sketches in compositionality. Cambridge University Press (2019)

[18] Freyd, P.J., Scedrov, A.: Categories, allegories. Elsevier (1990)

[19] Gu, T., Piedeleu, R., Zanasi, F.: A complete diagrammatic calculus for boolean satisfiability (2022), available at `piedeleu.com/publications/sat-algebra.pdf`

[20] Hasegawa, M.: Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In: Proceedings of the Third International Conference on Typed Lambda Calculi and Applications (TLCA). pp. 196–213. Springer (1997)

[21] Hinze, R.: Self-certifying railroad diagrams. In: International Conference on Mathematics of Program Construction (MPC). pp. 103–137. Springer (2019)

[22] Hoare, C., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene algebra. In: Proceedings of the 20th International Conference on Concurrency Theory (CONCUR). pp. 399–414. Springer (2009)

[23] Hyland, M., Schalk, A.: Glueing and orthogonality for models of linear logic. Theoretical Computer Science **294**(1-2), 183–231 (2003)

[24] Jacobs, B., Kissinger, A., Zanasi, F.: Causal inference by string diagram surgery. In: Proceedings of the 22nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS). pp. 313–329. Springer (2019)

[25] Joyal, A., Street, R., Verity, D.: Traced monoidal categories. In: Mathematical Proceedings of the Cambridge Philosophical Society. vol. 119, pp. 447–468. Cambridge University Press (1996)

[26] Kappé, T., Brunet, P., Silva, A., Zanasi, F.: Concurrent Kleene algebra: Free model and completeness. In: Proceedings of the 27th European Symposium on Programming (ESOP) (2018)

[27] Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. Journal of Pure and Applied Algebra **19**, 193–213 (1980)

[28] Kleene, S.C.: Representation of events in nerve nets and finite automata. Tech. rep., RAND PROJECT AIR FORCE SANTA MONICA CA (1951)

[29] Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2), 366–390 (1994)

[30] Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems (TOPLAS) **19**(3), 427–443 (1997)

[31] Kozen, D.C.: Automata and computability. Springer Science & Business Media (2012)

[32] Krob, D.: Complete systems of B-rational identities. Theoretical Computer Science **89**(2), 207–343 (1991)

[33] Lack, S.: Composing PROPs. Theory and Application of Categories **13**(9), 147–163 (2004)

[34] Lambek, J., Scott, P.J.: Introduction to higher-order categorical logic, vol. 7. Cambridge University Press (1988)

[35] Lawvere, F.W.: Functorial semantics of algebraic theories. Proceedings of the National Academy of Sciences of the United States of America **50**(5), 869 (1963)

[36] McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943)

[37] Moshier, A.M.: Coherence for categories of posets with applications. Topology, Algebra and Categories in Logic (TACL) p. 214 (2015)

[38] Piedeleu, R., Zanasi, F.: A string diagrammatic axiomatisation of finite-state automata. In: FoSSaCS. pp. 469–489 (2021)

[39] Pratt, V.: Action logic and pure induction. In: European Workshop on Logics in Artificial Intelligence. pp. 97–120. Springer (1990)

[40] Pratt, V.R.: Dynamic algebras and the nature of induction. In: Proceedings of the twelfth annual ACM symposium on Theory of Computing. pp. 22–28 (1980)

[41] Redko, V.N.: On defining relations for the algebra of regular events. Ukrainskii Matematicheskii Zhurnal **16**, 120–126 (1964)

[42] Selinger, P.: A survey of graphical languages for monoidal categories. Springer Lecture Notes in Physics **13**(813), 289–355 (2011)

[43] Smolka, S., Foster, N., Hsu, J., Kappé, T., Kozen, D., Silva, A.: Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. Proceedings of the 47th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) **4**, 1–28 (2020)

[44] Sobociński, P., Montanari, U., Melgratti, H., Bruni, R.: Connector algebras for C/E and P/T nets' interactions. Logical Methods in Computer Science **9** (2013)

[45] Stefanescu, G.: Network Algebra. Discrete Mathematics and Theoretical Computer Science, Springer London (2000)

[46] Thompson, K.: Programming techniques: Regular expression search algorithm. Communications of the ACM **11**(6), 419–422 (1968)

[47] Wirth, N.: The programming language pascal. Acta informatica **1**(1), 35–63 (1971)

[48] Zanasi, F.: Interacting Hopf Algebras: the theory of linear systems. Ph.D. thesis, Ecole Normale Supérieure de Lyon (2015)

## Appendix A. Background & Methodology

A.1. **Props, String Diagrams, and Symmetric Monoidal Theories.** We build on a line of research that has sought to give a formal treatment of graphical models of computation of varying expressive power within the unifying language of symmetric monoidal categories. More specifically, we rely on the notion of coloured product and permutations category (prop), a mathematical structure which generalises standard multisorted algebraic theories [12]. Formally, a *prop* is a strict symmetric monoidal category (SMC) whose objects are lists of a finite set of objects and where the monoidal product $\oplus$ on objects is given by list concatenation. Equivalently, it is a strict SMC whose objects are all monoidal products of a finite number of generating objects. *Prop morphisms* are strict symmetric monoidal functors that act as the identity on objects.

Following an established methodology, we define two props: Syn and Sem, for the syntax and semantics respectively. To guarantee a compositional interpretation, we require $\llbracket \cdot \rrbracket : \mathsf{Syn} \to \mathsf{Sem}$, the mapping of terms to their intended semantics, to be a prop morphism.

Typically, the syntactic prop Syn is freely generated from a *monoidal signature* $\Sigma = (O, M)$: a pair of a finite set of objects $O$ and a set $M$ of arrows $g : X \to Y$, where $X$ and $Y$ are lists of elements of $G$. In this case, we use the notation $\mathsf{P}_{\mathcal{S}}$ and Syn interchangeably. There are two ways of describing the arrows of the prop $\mathsf{P}_{\mathcal{S}}$ concretely. As terms of $(G^*, G^*)$-sorted syntax whose constants are elements of $\mathcal{S}$ and whose operations are the usual categorical composition $(-) ; (-) : \mathsf{Syn}(X, Y) \times \mathsf{Syn}(Y, Z) \to \mathsf{Syn}(X, Z)$ and the monoidal product $(-) \oplus (-) : \mathsf{Syn}(X_1, Y_1) \times \mathsf{Syn}(X_2, Y_2) \to \mathsf{Syn}(X_1 X_2, Y_1 Y_2)$, quotiented by the laws of SMCs. But this quotient is cumbersome and unintuitive to work with.

This is why, we will prefer a different representation. With their two forms of composition, monoidal categories admit a natural two-dimensional graphical notation of *string diagrams*. The idea is that an arrow $c : X \to Y$ of $\mathsf{P}_{\mathcal{S}}$ is better represented as a box with $|X|$ ordered wires labelled by the elements of $X$ on the left and $|Y|$ wires labelled by the elements of $Y$ on the left. We can compose these diagrams in two different ways: horizontally with ; by connecting the right wires of one diagram to the left wires of another when the types match, and vertically with $\oplus$ by juxtaposing two diagrams:

$$c \, ; d = \quad \overset{X}{\relbar}\boxed{c}\overset{Y}{\relbar}\boxed{d}\overset{Z}{\relbar} \qquad\qquad d_1 \oplus d_2 = \quad \begin{array}{c} \overset{X_1}{\relbar}\boxed{d_1}\overset{Y_1}{\relbar} \\ \overset{X_2}{\relbar}\boxed{d_2}\overset{Y_2}{\relbar} \end{array}$$

Thus, arrows of $\mathsf{P}_{\mathcal{S}}$ can be pictured as (directed acyclic) graphs whose nodes are labelled by elements of $\mathcal{S}$ and whose edges are identity $id_a : a \to a$, denoted as a plain wire $\relbar\!\!\relbar$ for generating object $a$. The symmetry $\mathcal{S}_{a,b} : a, b \to b, a$ is drawn as a wire crossing $\times$ which swaps the $a$-and $b$-wires, and the unit for $\oplus$, $id_0 : 0 \to 0$, as the empty diagram $\boxed{\phantom{x}}$ (we use 0 to denote the empty list). With this representation the laws of SMCs become diagrammatic tautologies.

Once we have defined $\llbracket \cdot \rrbracket : \mathsf{Syn} \to \mathsf{Sem}$, it is natural to look for equations to reason about semantic equality directly on the diagrams themselves. Given a set of equations $E$, i.e., a set containing pairs of arrows of the same type, we write $=_E$ for the smallest congruence w.r.t. the two composition operations ; and $\oplus$. We say that $=_E$ is *sound* if $c =_E d$ implies $\llbracket c \rrbracket = \llbracket d \rrbracket$. It is moreover *complete* when the converse implication also holds. We call a pair $(\mathcal{S}, E)$ a *symmetric monoidal theory* (SMT) and we can form the prop $\mathsf{P}_{\mathcal{S},E}$ obtained

by quotienting each homset of $\mathsf{P}_{\mathcal{S}}$ by $=_E$. There is then a prop morphism $q : \mathsf{P}_{\mathcal{S}} \to \mathsf{P}_{\mathcal{S},E}$ witnessing this quotient.

The reader familiar with categorical logic, may find it helpful to know that the concrete description above can be described in more abstract categorical terms, in line with Lawvere's account of algebraic theories [35]: signatures can be organised into a category and the free prop $\mathsf{P}_{\mathcal{S}}$ given as a monad structure over this category. Furthermore, the category of props and prop morphisms is equivalent to the category of algebras for this monad. Then, by standard abstract nonsense, the prop $\mathsf{P}_{\mathcal{S},E}$ and the quotient morphism $q$ arise as a coequaliser of free props. A detailed account of this presentation can be found in [3, Appendix A.2].

A.2. **Ordered Props and Symmetric Monoidal Inequality Theories.** Our semantic prop Sem often carries additional structure that we wish to lift to the syntax: relations or Boolean profunctors (which are relations satisfying an extra monotony condition) can be ordered by inclusion. The corresponding mathematical structure is that of an *ordered (or order-enriched) prop*, a prop whose homsets are also posets, with composition and monoidal product are monotone maps.

In the same way that props can be presented by SMTs, an ordered prop can be presented by *symmetric monoidal inequality theory* (SMIT). Formally, the data of a SMIT is the same as that of a SMT: a signature $\mathcal{S}$ and a set $I$ of pairs $c, d : X \to Y$ of $\mathsf{P}_{\mathcal{S}}$-arrows of the same type, that we now read as *inequalities $c \leq d$*.

As for plain props, we can construct an ordered prop from a SMIT by building the free prop $\mathsf{P}_{\mathcal{S}}$ and passing to a quotient $\mathsf{P}_{\mathcal{S},I}$. First, we build a preorder on each homset by closing $I$ under $\oplus$ and taking the reflexive and transitive closure of the resulting relation. Then, we obtain the free ordered prop $\mathsf{P}_{\mathcal{S},I}$ by quotienting the resulting preorder by imposing anti-symmetry.

An aside, for the reader comfortable with categorical logic: as for props and SMTs, we can give the concrete construction of this section a more abstract formulation, in terms of enriched category theory. The free order-enriched prop could be described as a monad over an order-enriched category of signatures, and the quotient prop $\mathsf{P}_{\mathcal{S},I}$ as a weighted-colimit. We will not need this characterisation here, so leave a detailed account (which we could not find in the literature) for future work.

SMITs subsume SMTs, since every SMT can be presented as a SMIT, by splitting each equation into two inequalities. As a result, in the main text, we only consider SMITs, referring to them simply as *theories*, and their defining inequalities as *axioms*. When referring to a sound and complete theory, we will also use the term *axiomatisation*, as is standard in the literature.

The situation for a sound and complete theory is summarised in the commutative diagram below:

$$\mathsf{Syn} = P_{\mathcal{S}} \xrightarrow{\quad \llbracket \cdot \rrbracket \quad} \mathsf{Sem}$$

with $q$ and $s$ maps through $P_{\mathcal{S},I}$.

Soundness simply means that $\llbracket \cdot \rrbracket$ factors as $s \circ q$ through $\mathsf{P}_{\mathcal{S},E}$ and completeness means that $s$ is a faithful prop morphism.