A String Diagrammatic Axiomatisation of Finite-State Automata

Robin Piedeleu(✉) and Fabio Zanasi

University College London, London, UK,
{r.piedeleu, 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— a result which is provably impossible for the one-dimensional syntax of regular expressions. Second, the Kleene star is a derived concept, as it can be decomposed into more primitive algebraic blocks.

Keywords: string diagrams · finite-state automata · symmetric monoidal category · complete axiomatisation

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 [25] introduced regular expressions to give finite-state automata an algebraic presentation, motivated by the study of (biological) neural networks [31]. They are the terms freely generated by the following grammar:

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

Equational properties of regular expressions were studied by Conway [14] 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 [35]. Instead, there are purely equational infinitary axiomatisations [28,4] and Kozen's finitary implicational theory [26].

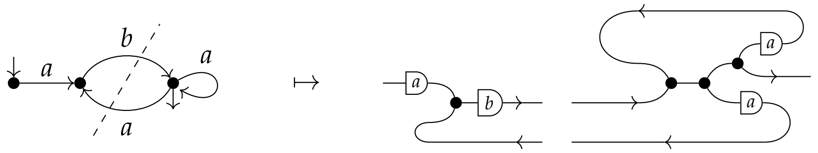
Since then, much research has been devoted to extending Kleene algebra with operations capturing richer patterns of behaviour, useful in program verification. Examples include conditional branching (Kleene algebra with tests [27], and its recent guarded version [37]), concurrent computation (CKA [19,23]), and specification of message-passing behaviour in networks (NetKAT [1]).

The meta-theory of the formalisms above essentially rests on the same three ingredients: (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.

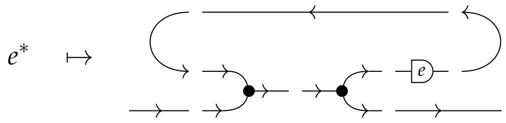
In this paper, we open up a direct path from (1) to (3). Instead of thinking of automata as a combinatorial model, we formalise them as a bona-fide (two-dimensional) syntax, using the well-established mathematical theory of *string diagrams* and monoidal categories [36]. This approach lets us axiomatise 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. Intriguingly, 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:

 (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):

 (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 [12]. 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.

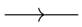
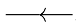
We should also note that this is not the first time that automata and regular languages are recast into a categorical mould. The *iteration theories* [5] of Bloom and Ésik, *sharing graphs* [17] of Hasegawa or *network algebras* [39] of Stefanescu are all categorical frameworks designed to reason about 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 [22]. 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.

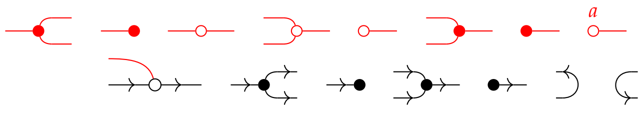
A second difference is methodological: our syntax (4) does not feature any primitive for iteration or recursion. In particular, the star is a derived concept, in the sense that it is decomposable into more elementary operations (3). Categorically, our starting point is a compact-closed rather than traced category.

We elaborate on the relation between ours and existing work in Section 6. Omitted proofs can be found in [33].

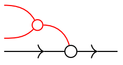
2 Syntax and semantics

Syntax. We fix an alphabet Σ of letters $a \in \Sigma$. We call Aut_Σ the symmetric strict monoidal category freely generated by the following objects and morphisms:

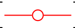
- three generating objects ▶ ('action'), ▶ ('right') and ◀ ('left') with their identity morphisms depicted respectively as ,  and .
- the following generating morphisms, depicted as *string diagrams* [36]:

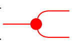


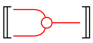
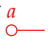
$(a \in \Sigma)$ (4)

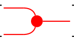
Freely generating Aut_Σ from these data (usually called a *symmetric monoidal theory* [42,11]) means that morphisms of Aut_Σ will be the string diagrams obtained by pasting together (by sequential composition and monoidal product in Aut_Σ) the basic components in (4), and then quotienting by the laws of symmetric monoidal categories. For instance, (3) is a morphism of Aut_Σ of type ▶→▶, and  is one of type ▶▶ ▶ → ▶.

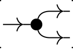
Semantics. We first define the semantics for string diagrams simply as a function, and then discuss how to extend it to a functor from Aut_Σ to another category. Our interpretation maps generating morphisms to relations between regular expressions and languages over Σ:

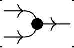
$[\![$ $]\!] = \{((e,e) \mid e \in \mathsf{RegExp}\}$ $[\![$  $]\!] = \{(e,e^*) \mid e \in \mathsf{RegExp}\}$

$[\![$  $]\!] = \{(e,(e,e)) \mid e \in \mathsf{RegExp}\}$ $[\![$ $]\!] = \{(e,\bullet) \mid e \in \mathsf{RegExp}\}$

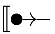
$[\![$  $]\!] = \{((e,f),ef) \mid e,f \in \mathsf{RegExp}\}$ $[\![$ $]\!] = \{(\bullet,1)\}$ $[\![$  $]\!] = \{(\bullet,a)\}$

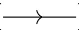
$[\![$  $]\!] = \{((e,f),e+f) \mid e,f \in \mathsf{RegExp}\}$ $[\![$ $]\!] = \{(\bullet,0)\}$

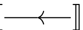
$[\![$  $]\!] = \{(L,(K_1,K_2)) \mid L \subseteq K_i,\ i = 1,2 \text{ and } L,K_1,K_2 \subseteq \Sigma^\star\}$

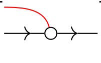
$[\![$  $]\!] = \{((L_1,L_2),K) \mid L_i \subseteq K,\ i = 1,2 \text{ and } L_1,L_2,K \subseteq \Sigma^\star\}$

$[\![$ $]\!] = \{(L,\bullet) \mid L \subseteq \Sigma^\star\}$ $[\![$ $]\!] = \{(\bullet,(L,K)) \mid L \subseteq K \mid L,K \subseteq \Sigma^\star\}$

$[\![$  $]\!] = \{(\bullet,K) \mid K \subseteq \Sigma^\star\}$ $[\![$ $]\!] = \{((L,K),\bullet) \mid K \subseteq L \mid L,K \subseteq \Sigma^\star\}$

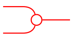
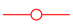
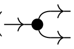
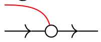
$[\![$  $]\!] = \{((L,K),L \subseteq K) \mid L,K \subseteq \Sigma^\star\}$

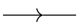
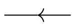
$[\![$  $]\!] = \{((L,K),K \subseteq L) \mid L,K \subseteq \Sigma^\star\}$

$[\![$  $]\!] = \{((e,L),K) \mid L\,[\![e]\!]_R \subseteq K \text{ and } e \in \mathsf{RegExp}, L,K \subseteq \Sigma^\star\}$ (5)

In (5), the semantics $[\![e]\!]_R \in 2^{A^*}$ of a regular expression $e \in \mathsf{RegExp}$ is defined inductively on e (see (1)), in the standard way:

$$[\![e+f]\!]_R = [\![e]\!]_R \cup [\![f]\!]_R \qquad [\![ef]\!]_R = \{vw \mid v \in [\![e]\!]_R\,, w \in [\![f]\!]_R\}$$
$$[\![1]\!]_R = \{\varepsilon\} \qquad [\![0]\!]_R = \varnothing \qquad [\![a]\!]_R = \{a\} \qquad [\![e^*]\!]_R = \bigcup_{n\in\mathbb{N}} [\![e^n]\!]_R$$

where $e^{n+1} := ee^n$ and $e^0 := 1$. The semantics highlights the different roles played by red[1] and black generators. In a nutshell, red generators stand for regular expressions (the sum, is 0,  the product, is 1,  the Kleene star, and the letters of Σ), and black generators for operations on the set of languages ( is copy, is delete, and feed back outputs into inputs, in a way made more precise later). These two perspectives, which are usually merged, are kept distinct in our approach and only allowed to communicate via  , which represents the product action of regular expressions (the red wire) on languages via concatenation on the right.

In order for this mapping to be functorial from Aut_Σ, we now introduce a suitable target semantic category. Interestingly, this will not be the category Rel of sets and relations: indeed, the identity morphisms  and  are not interpreted as identities of Rel. Instead, the semantic domain will be the category $\mathsf{Prof}_{\mathbb{B}}$ of *Boolean(-enriched) profunctors* [15] (also called in the literature relational profunctors [20] or weakening relations [32]).

Definition 1. *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* $(x',y') \in R$*.*

[1] The reader with a greyscale version of the paper should see light grey generators instead.

Preorders and Boolean profunctors form a symmetric monoidal category $\mathsf{Prof}_{\mathbb{B}}$ *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.*

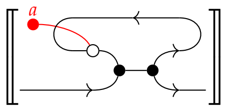
The rich features of our diagrammatic language are reflected in the profunctor interpretation. Indeed, the order relation is built into the wires $\rightarrow$ and $\leftarrow$. The two possible directions represent the identities on the ordered set of languages and the same set with the reversed order, respectively. The additional red wire $-$ represents the set RegExp of regular expressions, with *equality* as the associated order relation.[2] It is clear that all monochromatic generators satisfy the condition of Definition 1. Similarly, the action generator $\rightarrow\!\circ\!\rightarrow$ is a Boolean profunctor: if $((e, L), K)$ are such that $L \llbracket e \rrbracket_R \subseteq K$ and $L' \subseteq L$, $K \subseteq K'$ then we have $L' \llbracket e \rrbracket_R \subseteq L \llbracket e \rrbracket_R \subseteq K \subseteq K'$ by monotony of the product of languages. We can conclude that

Proposition 1. $\llbracket \cdot \rrbracket$ *defines a symmetric monoidal functor of type* $\mathsf{Aut}_\Sigma \to \mathsf{Prof}_{\mathbb{B}}$.

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

$$\llbracket -\boxed{c}-\boxed{d}- \rrbracket = \left\{ (L, K) \mid \exists M\, (L, M) \in \llbracket -\boxed{c}- \rrbracket, (M, K) \in \llbracket -\boxed{d}- \rrbracket \right\}$$
$$\left\llbracket \begin{matrix} -\boxed{c_1}- \\ -\boxed{c_2}- \end{matrix} \right\rrbracket = \left\{ ((L_1, L_2), (K_1, K_2)) \mid (L_i, K_i) \in \llbracket -\boxed{c_i}- \rrbracket, i = 1, 2 \right\}$$

Example 1. 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^*. We assign variable names to each wire: O to the top wire of the feedback loop, N to the output wire of the action node, and M to the middle wire joining $\rightrightarrows\!\bullet\!\rightarrow$ to $\rightarrow\!\bullet\!\rightrightarrows$ so that we can compute:



$$\begin{aligned} &= \{(L, K) \mid \exists M, N, O,\ L, N \subseteq M,\ O \llbracket a \rrbracket_R \subseteq N,\ M \subseteq O, K\} \\ &= \{(L, K) \mid \exists N, O,\ L, N \subseteq O,\ L, N \subseteq K\ Oa \subseteq N\} \\ &= \{(L, K) \mid \exists O,\ Oa \subseteq O,\ L \subseteq O,\ L, O \subseteq K\}. \end{aligned}$$

Call this diagram d. Since $Oa \subseteq O$ and $L \subseteq O$ is equivalent to $L \cup Oa \subseteq O$, $\llbracket d \rrbracket = \{(L, K) \mid \exists O \text{ s.t. } L \cup Oa \subseteq O,\ L, O \subseteq K\}$. Finally, by Arden's lemma [2], La^* is the *least* solution of the language inequality $L \cup Xa \subseteq X$; thus $\llbracket d \rrbracket = \{(L, K) \mid \exists O \text{ s.t. } La^* \subseteq O,\ L, O \subseteq K\} = \{(L, K) \mid\ La^* \subseteq K\}$.

3 Equational theory

In Figure 1 we introduce $=_{KDA}$, the (finite) equational theory of *Kleene Diagram Algebra*, on Aut_Σ. It will be later shown to be *complete* for the given semantics. We explain some salient features of $=_{KDA}$ below.

[2] Note that we can always consider any set with equality as a poset and that, therefore, Rel is a subcategory of $\mathsf{Prof}_{\mathbb{B}}$, but not vice-versa, for the simple reason that the identity relation of an arbitrary poset in $\mathsf{Prof}_{\mathbb{B}}$ is not mapped to the identity relation in Rel.

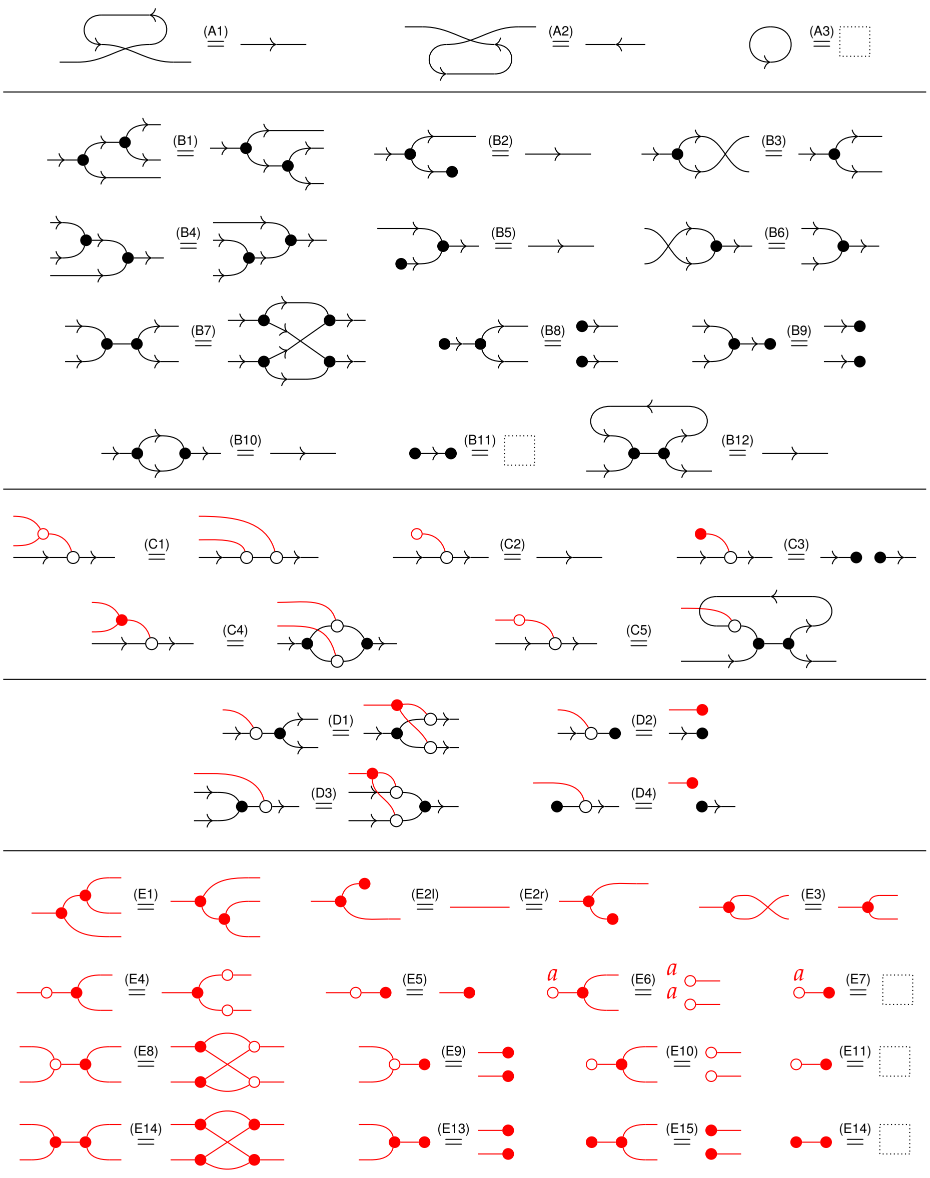
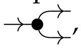
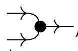
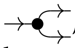
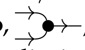
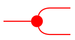
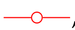
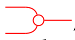
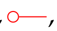
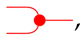
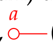
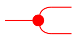


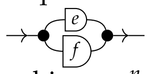
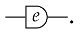
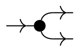
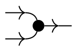
Fig. 1. Equational theory $=_{KDA}$ of Kleene Diagram Algebra.

- (A1)-(A2) relate and , allowing us to bend and straighten wires at will. This makes the full subcategory of Aut_Σ on ▶ and ◀, modulo (A1)-(A2), *compact closed* [24]. (A3) allows us to eliminate isolated loops. Note that the whole category is not compact closed because ▶ has no dual.
- The B block states that , forms a cocommutative comonoid (B1)-(B3), while , form a commutative monoid (B4)-(B6). Moreover, , , , form an idempotent bimonoid (B7)-(B11). (B12) allows us to eliminate trivial feedback loops.
- The C block axiomatises the action of regular expressions on languages. These laws mimic the usual definition of the action of a semiring on a set, except for (C5) which is novel and captures the interaction with the Kleene star. Here lies a distinctive feature of our theory: the behaviour of the star is derived from its decomposition as the feedback loop on the right of (C5).
- The D block forces the action to be a comonoid ((D1)-(D2)) and monoid ((D1)-(D2)) homomorphism.
- The E block axiomatises the purely red fragment. Remarkably, these axioms do not describe any of the actual Kleene algebra structure: they just state that  and form a commutative comonoid ((E1)-(E3)) and that all other red generators are comonoid homomorphisms ((E4)-(E15)). This means that the red fragment is actually the *free* (cartesian) algebraic theory (*cf.* [42,11]) on generators , , , , ,  ($a \in \Sigma$), where the remaining generators  and act as copy and discard of variables.

Let $=_{KDA}$ be the smallest equational theory containing all equations in Fig. 1. Their *soundness* for the chosen semantics is not difficult to show and, for space reasons, we omit the proof. We now state our *completeness* result, whose proof will be discussed in Section 5.

Theorem 1 (Completeness). *For morphisms d, e in* Aut_Σ *, $d =_{KDA} e$ iff $[\![d]\!] = [\![e]\!]$.*

Remark 1. In the usual approach to the theory of regular languages (e.g. [26]), a completeness result like Theorem 1 is typically proven by first defining a class of models for the algebraic theory, and showing that the standard semantics constitutes the initial/free model. Our proof is different in flavour, but equivalent: taking advantage of the categorical formulation of our diagrammatic syntax and its semantics, we construct an equivalence of categories between our model and the diagrams quotiented by the equations of KDA.

Remark 2. Some axiomatisations of Kleene algebra use a partial order between terms, which can be defined from the idempotent monoid structure: $f \leq e$ iff $e + f = e$. At the semantic level, it corresponds to inclusion of languages. Similarly, using the idempotent bimonoid structure of our equational theory, we can define a partial order on ▶→▶ diagrams: $f \leq e$ iff  $=$ . This partial order structure can also be extended to all morphisms $▶^n \to ▶^m$ by using the vertical composition of n copies of  and m copies of  instead.

Remark 3. There are no specific equations relating the atomic actions $\overset{a}{\circ}\!\!-\!\!-$ ($a \in \Sigma$). This is because, as we study automata, we are interested in the *free* monoid Σ^* over Σ. However, nothing would prevent us from modelling other structures. Free commutative monoids (powers of $\mathbb{N}$), whose rational subsets correspond to semilinear sets [14, Chapter 11] would be of particular interest.

4 Encoding regular expressions and automata

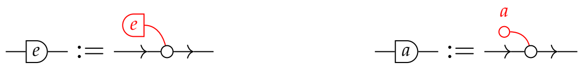
A major appeal of our approach is that both regular expressions and automata can be uniformly represented in the graphical language of string diagrams, and the translation of one into the other becomes an equational derivation in $=_{KDA}$. In fact, we will see there is a close resemblance between automata and the shape of the string diagrams interpreting them — the main difference being that string diagrams are *composable* structures.

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).

In a sense, regular expressions are already part of the graphical syntax, as the red generators: for any regular expression e, one may always construct a 'red' string diagram $\boxed{e}\!\!-\!\!-: 0 \to \blacktriangleright$ such that $[\![\, \boxed{e}\!\!-\!\!-]\!] = \{(\bullet, e)\}$. However, these alone are meaningless, since their image under the semantics is simply the free term algebra RegExp (see (7)) . They acquire meaning as they *act* on the set of languages over Σ, represented by the black wire.

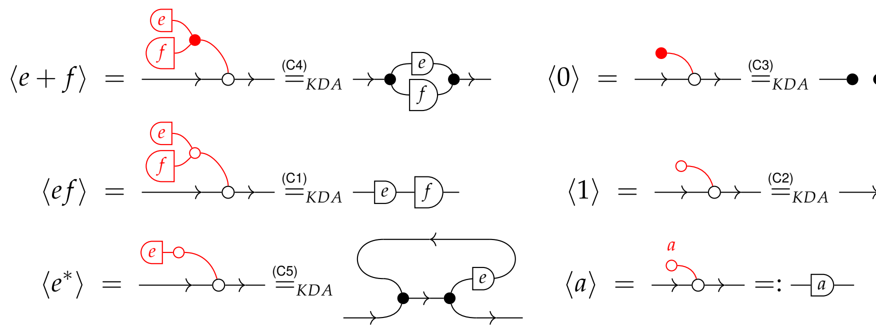
4.1 From regular expressions to string diagrams

To define these encodings, it is convenient to introduce the following syntactic sugar. We will write $-\!\boxed{e}\!-$ for the composite of $\boxed{e}\!\!-\!\!-$ with the action, as defined below left, with the particular case of a letter $a \in \Sigma$ on the right:



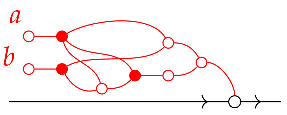
(6)

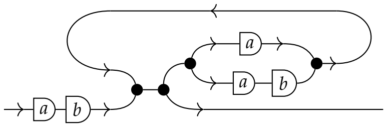
Using this action, we can inductively define an encoding $\langle - \rangle$ of regular expressions into string diagrams of Aut_Σ, as the rightmost diagram for each expression below:



(7)

For example, $\langle ab(a+ab)^{*}\rangle \ =$



$=_{KDA}$


(8)

As expected, the translation preserves the language interpretation of regular expressions in a sense that the following proposition makes precise.

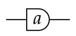
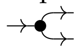
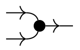
Proposition 2. *For any regular expression e, $[\![\langle e\rangle]\!] = \{(L,K) \mid [\![e]\!]_R\, L \subseteq K\}$.*

4.2 From automata to string diagrams...

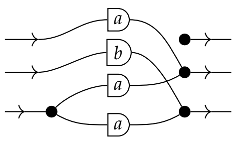
Example (8) suggests that the string diagram $\langle e\rangle$ corresponding to a regular expression e looks a lot like a nondeterministic finite-state automaton (NFA) for e. In fact, the translation $\langle - \rangle$ can be seen as the diagrammatic counterpart of Thompson's construction [40] that builds an NFA from a regular expression.

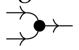
We can generalise the encoding of regular expressions and translate NFA directly into string diagrams, in at least two ways. The first is to encode an NFA as the diagrammatic counterpart of its transition relation. The second is to translate directly its graph representation into the diagrammatic syntax.

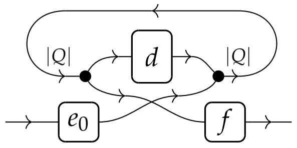
Encoding the transition relation. This is a simple variant of the translation of matrices over semirings that has appeared in several places in the literature [29,42].

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 δ as a string diagram d with $|Q|$ incoming wires on the left and $|Q|$ outgoing wires on the right.The left jth port of d is connected to the ith port on the right through an  whenever $(q_i, a, q_j) \in \delta$. To accommodate nondeterminism, when the same two ports are connected by several different letters of Σ, we join these using  and  . 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 δ such that $(q_i, a, q_j) \in \delta$ for any a, the two corresponding ports are disconnected.

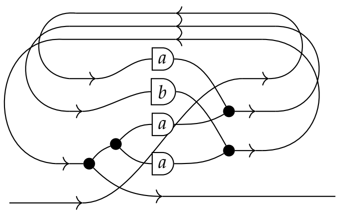
For example, the transition relation of an NFA with three states and $\delta = \{((q_0,a,q_1),(q_1,b,q_2),(q_2,a,q_1),(q_2,a,q_2))\}$ (disregarding the initial and accepting states for the moment) is depicted on the right. Conversely, given such a diagram, we can recover δ by collecting Σ-weighted paths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

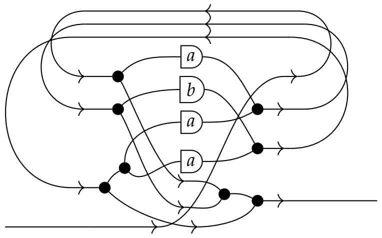
$d =$ 

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  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 δ, we obtain the string diagram on the right, where d is the diagrammatic counterpart of δ as defined above, e_0 is the injection of a single wire as the first amongst $|Q|$ wires, and f deletes all wires that are not associated to states in F with, and applies to merge them into a single outgoing wire.



For example, if A with δ as above has initial state q_0 and accepting state $\{q_2\}$, we get the diagram below left; instead, if all states are accepting, we obtain the diagram below right:

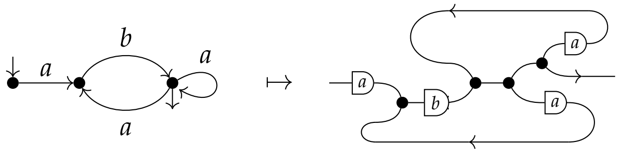




The correctness of this simple translation is justified by a semantic correspondence between the language recognised by a given NFA A and the denotation of the corresponding string diagram.

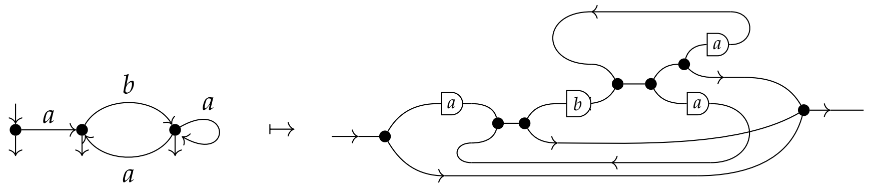
Proposition 3. *Given an NFA A which recognises the language L, let d_A be its associated string diagram, constructed as above. Then $[\![d_A]\!] = \{(K, K') \mid LK \subseteq K'\}$.*

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



(9)

Note the initial state of the automaton corresponds to the left interface of the string diagram, and the accepting state to the right interface. As before, when there are multiple accepting states, they all connect to a single right interface, via. For example, if we make all states accepting in the automaton above, we get the following diagrammatic representation:



4.3 ...and back

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 string diagram $d\colon\ \blacktriangleright \to \blacktriangleright$, such that the language the automaton recognises matches the denotation of d.

In order to phrase this correspondence formally, we need to introduce some terminology. We call *left-to-right* those string diagrams whose domain and codomain 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, (9) is a left-to-right diagram $\blacktriangleright \to \blacktriangleright$.

A string diagram d is *atomic* if the only red generators occurring in d are of the form $\overset{a}{\circ}\!\!-$. By *unfolding* all red components $\boxed{e}\!-$ in any left-to-right diagram, using axioms (C1)-(C5), we can prove the following statement.

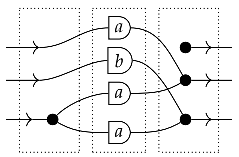
Proposition 4. *Any left-to-right diagram is $=_{KDA}$-equivalent to an atomic one.*

For instance, the string diagram on the left of (8) is $=_{KDA}$-equivalent to the atomic one on the right.

We call *block* of a certain subset of generators a vertical composite of these generators followed by some permutations of the wires.

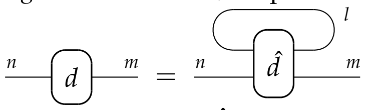
Definition 2. *A* matrix-diagram *(resp.* generalised matrix-diagram*) is a left-to-right diagram that factors as a block of* $\to\!\bullet\!\!<$, $\to\!\bullet$*, followed by a block of* $-\!\boxed{a}\!-$ *for* $a \in \Sigma$ *(resp.* $-\!\boxed{e}\!-$ *for* $e \in \mathsf{RegExp}$*) and finally, a block of* $>\!\!\bullet\!\to\!-$, $\bullet\!\to\!-$.

To each matrix-diagram d we can associate a unique transition relation δ by gathering paths from each input to each output: $(q_i, a, q_j) \in \delta$ if there is $-\!\boxed{a}\!-$ joining the ith input to the jth output.

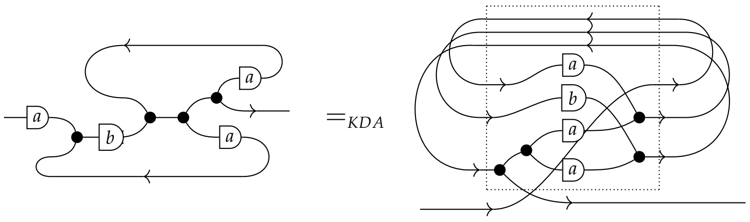


A transition relation is *ϵ-free* if it does not contain the empty word. It is *deterministic* if it is ϵ-free and, for each i and each $a \in \Sigma$ there is at most one j such that $(q_i, a, q_j) \in \delta$. We will apply these terms to matrix-diagrams and the associated transition relation interchangeably. The example of Section 4.2 above, with the three blocks highlighted, is a matrix-diagram. It is ϵ-free but not deterministic since there are two a-labelled transitions starting from the third input.

Given a matrix-diagram $d : \blacktriangleright^{l+n} \to \blacktriangleright^{p+m}$, we will write d_{ij}, with $i = l, n$ and $j = p, m$, for the subdiagrams corresponding to the appropriate submatrices.

Definition 3. *For any left-to-right diagram $d : \blacktriangleright^n \to \blacktriangleright^m$, a* representation *is a matrix-diagram $\hat{d} : \blacktriangleright^{l+n} \to \blacktriangleright^{l+m}$, such that*  *and $\hat{d}_{ll}$, $\hat{d}_{nl}$ are ϵ-free. It is a* deterministic representation *if moreover $\hat{d}_{ll}$ is deterministic.*

For example, given the string diagram below on the left, the one on the right is a representation for it, whose highlighted matrix-diagram is the same as above.



(10)

We will refer to the associated matrix-diagram $\hat{d}$ as the *transition matrix* of a given representation. From a $\blacktriangleright \to \blacktriangleright$ diagram with representation $\hat{d} : \blacktriangleright^{l+1} \to \blacktriangleright^{l+1}$ we can construct an NFA from its transition matrix $\hat{d}$ as follows:

– its state set is $Q = \{q_1, \dots, q_l\}$, i.e., there is one state for each wire of $\hat{d}_{ll}$;
– its transition relation built from $\hat{d}_{ll}$ as described above;
– its initial states Q_0 are those q_i for which there exists an index j such that the ijth coefficient of $\hat{d}_{1l}$ is non-zero (and therefore ϵ);
– its final states F are those q_j for which there exists an index i such that the ijth coefficient of $\hat{d}_{l1}$ is non-zero (and therefore ϵ);

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 3.

Proposition 5. *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 states that a representation can be extracted from any string diagram.

Proposition 6. *Any left-to-right diagram has a representation.*

We established a correspondence between $\blacktriangleright \to \blacktriangleright$ diagrams and automata. What about arbitrary left-to-right diagrams $\blacktriangleright^n \to \blacktriangleright^m$? To characterise the precise relationship between our syntax and regular expressions we can prove a *Kleene theorem* for Aut_Σ. Recall, from Definition 2 that a *generalised matrix-diagram* is the diagrammatic counterpart of a matrix whose coefficients are regular expressions. It turns out that every left-to-right diagram can be put in this form.

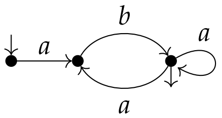
Proposition 7 (Kleene's for Aut_Σ). *Any left-to-right diagram is equal to a generalised matrix diagram.*

As a result, the semantics of a given $\blacktriangleright^n \to \blacktriangleright^m$ diagram is fully characterised by an $m \times n$ array of regular languages.

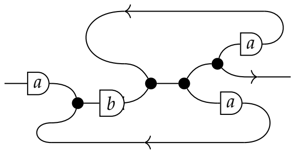
4.4 Interlude: from regular to context-free languages

It is worth pointing out how a simple modification of the diagrammatic syntax takes us one notch up the Chomsky hierarchy, leaving the realm of regular languages for that of context-free grammars and languages.

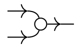
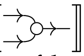
Our syntax allows to specify systems of language equations of the form $aX \subseteq Y$. In this context, feedback loops can be interpreted as fixed-points. For example, the automaton below left, and its corresponding string diagram, below right, translate to the system of equations at the center:



$$\mapsto \begin{cases} \epsilon \subseteq X_0 \\ X_0 a \subseteq X_1 \\ X_1 b \subseteq X_2 \\ X_2 a \subseteq X_1 \\ X_2 a \subseteq X_2 \end{cases} \leftarrow\!\shortmid \tag{11}$$

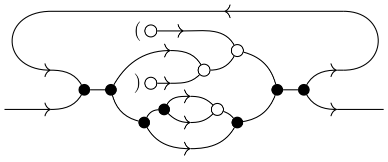


This translation can be obtained by simply labelling each state with a variable and adding one inequality of the form $X_i a \subseteq X_j$ for each a-transition from state i to state j. The system we obtain corresponds very closely to the $[\![-]\!]$-semantics of the associated string diagram.

The distinction between red and black wires can be understood as a type discipline that only allows linear uses of the product of languages. It is legitimate and enlightening to ask what would happen if we forgot about red wires and interpreted the action directly as the product. We would replace the action by a new generator  with semantics $[\![$  $]\!] = \{((M, L), K) \mid ML \subseteq K\}$.

This would allow us to specify systems of language equations with unrestricted uses of the product on the left of inclusions, e.g. $UVW \subseteq X$. Equations of this form are similar to the production rules (e.g. $X \to UVW$) of context-free grammars and it is well-known that the least solutions of this class of systems are precisely *context-free* languages [14, Chapter 10].

For example we could encode the language $X \to XX \mid (X) \mid \epsilon$ of properly matched parentheses as least solution of the system $\epsilon \subseteq X, (X) \subseteq X, XX \subseteq X$ which gives the diagram displayed on the right.



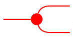
5 Completeness and Determinisation

This section is devoted to prove our completeness result, Theorem 1. 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). We achieve it by implementing Brzozowski's algorithm [12] through diagrammatic equational reasoning. The proof proceeds in three distinct steps.

1. We first show (Section 5.1) how to *determinise* (the representation of) a diagram: this step consists in eliminating all subdiagrams that correspond to nondeterministic transitions in the associated automaton.
2. We use the previous step to implement a *minimisation* procedure (Section 5.2) 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 $=_{KDA}$.
3. 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 $=_{KDA}$.

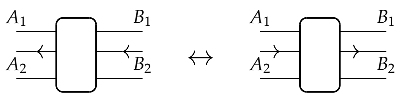
We will now write equations in $=_{KDA}$ simply as $=$ to simplify notation and say that diagrams c and d are *equal* when $c =_{KDA} d$.

First, we use the symmetries of the equational theory to make simplifying assumptions about the diagrams to consider in the completeness proof.

A few simplifying assumptions. Without loss of generality, the proof we give is restricted to string diagrams with no ▶ in their domain as well as in their codomain. This is simply a matter of convenience: the same proof would work for more general diagrams, that may contain ▶ in their (co)domain, at the cost of significantly cluttering diagrams. Henceforth, one can simply think of the labels for the action $-\boxed{x}-$ as uniquely identifying one open red wire in a diagram. With this convention, two or more occurrences of the same x in a diagram can be seen as connected to the same red wire on the left, via . That we can safely do so is a consequence of the completeness of $=_{KDA}$ restricted to the monochromatic red fragment, itself a consequence of [11, Theorem 6.1].

Arbitrary objects in Aut_Σ are lists of the three generating objects. We have already motivated focusing on string diagrams with no open red wires so that the objects we care about are lists of ▶ and ◀. The following proposition implies that, without loss of generality, for the proof of completeness we can restrict further to left-to-right diagrams (Section 4.2).

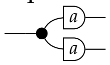
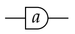
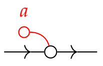
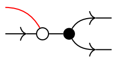
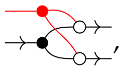
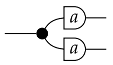
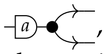
Proposition 8. *There is a natural bijection between sets of string diagrams of the form*



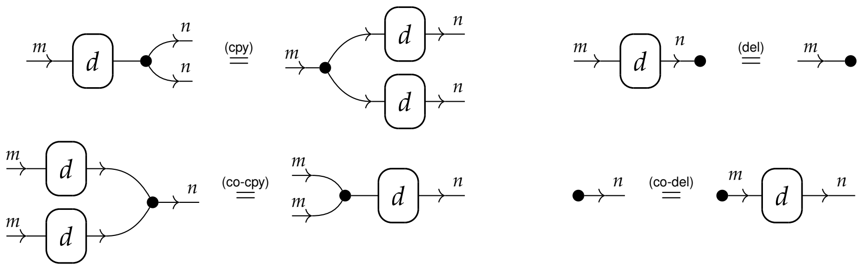
where A_i, B_i represent lists of ▶ and ◀.

Proposition 8 tell us that we can always bend the 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.

5.1 Determinisation

In diagrammatic terms, a nondeterministic transition of the automaton associated to (a representation of) a given diagram, corresponds to a subdiagram of the form  for some $a \in \Sigma$. Clearly, using the definition of  :=  in (6) and the axiom  $\overset{\text{(D1)}}{=}$ , we have  = , which will prove to be the engine of our determinisation procedure, along with the fact that any red expression can be copied and deleted. The next two theorems generalise the ability to copy and delete to arbitrary left-to-right diagrams.

Theorem 2. *For any left-to-right diagram $d : \blacktriangleright^m \to \blacktriangleright^n$, we have*



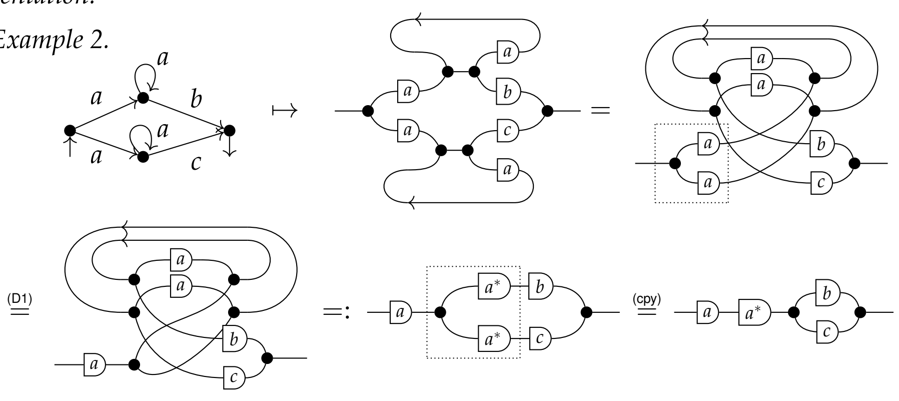
For $d : \blacktriangleright^m \to \blacktriangleright^n$, let d_{ij} be the string diagram of type $\blacktriangleright \to \blacktriangleright$ obtained by composing every input with ●→— except the ith one, and every output with →● except the jth one. Theorem 2 implies that string diagrams are fully characterised by their $\blacktriangleright \to \blacktriangleright$ subdiagrams.

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

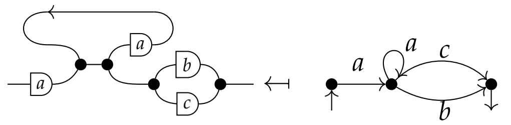
Thus, we can restrict our focus further to left-to-right $\blacktriangleright \to \blacktriangleright$ diagrams, without loss of generality. We are now able to devise a determinisation procedure for representation of diagrams, which we illustrate below on a simple example.

Proposition 9 (Determinisation). *Any diagram $\blacktriangleright \to \blacktriangleright$ has a deterministic representation.*

Example 2.



:=



Dealing with useless states. Notice that our deterministic form is *partial* and that the determinisation procedure disregards *useless states*, i.e., parts of a string diagram that do not reach an output wire. None of these contribute to the semantics of the diagram and can be safely eliminated using Theorem 2 (del)-(co-del).

5.2 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 looking at string diagrams differently and showing 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 (Theorem 1 (Completeness)). We have a procedure to show that, if $[\![d]\!] = [\![e]\!]$, then there exists a string diagram f in normal form such that $d = f = e$. This normal form is the diagrammatic counterpart of the *minimal* automaton associated to d and e. In our setting, it is the deterministic representation equal to d and e 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 6. 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 laws all hold in reverse. 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 Proposition 9, we can obtain a deterministic representation for the co-deterministic representation of the previous step. The result is the desired minimal representation and normal form.

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 or railroad diagrams* to visualise regular expressions and context-free grammars [41]. These diagrams resemble our 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 [18]. 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 [5]. 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* (only the full subcategory over $\blacktriangleright$ and $\blacktriangleleft$ to be precise), a property that is incompatible with the existence of categorical products (any category that has both collapses to a preorder [30]). Nevertheless, the subcategory of left-to-right diagrams (Section 4.2) is a (matrix) iteration theory [6], a structure that Bloom and Ésik have used to give an (infinitary) axiomatisation of regular languages [4].

Similarly, Stefanescu's work on *network algebra* provides a unified algebraic treatment of various types of networks, including finite-state automata [39]. 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 (3).

Note that the compact closed subcategory in this paper can be recovered from the traced monoidal category of left-to-right diagrams, via the *Int construction* [22]. 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 (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.

Finally, the idea of treating regular expressions as a free structure acting on a second algebraic structure also appeared in Pratt's *dynamic algebras*, which axiomatise the propositional fragment of dynamic modal logic [34]. Like our formalism, the variety of dynamic algebras is finitely-based. But they assume more structure: the second algebraic structure is a Boolean algebra.

In all the formalisms we have mentioned, the difficulty typically lies in capturing the behaviour of iteration—whether as the star in Kleene algebra [26,4], or a trace operator [5] in iteration theory and network algebra [39]. 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 [26] 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 other axioms. (b) is more subtle, but can be seen as a consequence of (D1)-(D4) axioms. These allows us to (co)copy and (co)delete arbitrary diagrams (Theorem 2) and we conjecture that this is what forces the star to be a single definite value, not just any fixed-point, but the least one. Making this statement precise is the subject of future work.

The difficulty in capturing the behaviour of fixed-points is also the reason why we decided to work with an additional red wire, to encode the action of regular expressions on the set of languages—without it, global (co)copying and (co)deleting (Theorem 2) cannot be reduced to the local (D1)-(D4) axioms. There is another route, that leads to an infinitary axiomatisation: we could dispense with the red generators altogether and take $-\boxed{a}-$ (for $a \in \Sigma$) as primitive instead, with global axioms to (co)copy and (co)delete arbitrary diagrams. This would pave the way for a reformulation of our work in the context of iteration (matrix) theories, where the ability to (co)copy and (co)delete arbitrary expressions is already built-in. We leave this for future work.

There is an intriguing parallel between our case study and the positive fragment of relation algebra (also known as allegories [16]). Indeed, allegories, like Kleene algebra, do not admit a finite axiomatisation [16]. 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 [9]. It seems like the greater generality of the monoidal setting—algebraic theories correspond precisely to the particular case of cartesian monoidal categories [11]—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) [19,23] 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. [7,38]), and more generally about resource exchange between processes (see e.g. [10,13,21,3,8]). 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 (3)) and there is only one specific axiom (C5) governing its behaviour. 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.

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.C., Fong, B.: A compositional framework for passive linear networks. Theory & Applications of Categories **33** (2018)
4. Bloom, S.L., Ésik, Z.: Equational axioms for regular sets. Mathematical structures in computer science **3**(1), 1–24 (1993)
5. Bloom, S.L., Ésik, Z.: Iteration theories. Springer (1993)
6. Bloom, S.L., Ésik, Z.: Matrix and matricial iteration theories. Journal of Computer and System Sciences **46**(3), 381–439 (1993)
7. 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)
8. 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)
9. Bonchi, F., Seeber, J., Sobocinski, P.: Graphical conjunctive queries. In: 27th Annual EACSL Conference Computer Science Logic, (CSL). vol. 119 (2018)
10. Bonchi, F., Sobociński, P., Zanasi, F.: The calculus of signal flow diagrams I: linear relations on streams. Information and Computation **252**, 2–29 (2017)
11. Bonchi, F., Sobociński, P., Zanasi, F.: Deconstructing Lawvere with distributive laws. Journal of logical and algebraic methods in programming **95**, 128–146 (2018)
12. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. Mathematical theory of Automata **12**(6), 529–561 (1962)
13. Coecke, B., Kissinger, A.: Picturing Quantum Processes - A first course in Quantum Theory and Diagrammatic Reasoning. Cambridge University Press (2017)
14. Conway, J.H.: Regular algebra and finite machines. Courier Corporation (2012)
15. Fong, B., Spivak, D.I.: Seven sketches in compositionality: An invitation to applied category theory. arXiv:1803.05316 (2018)
16. Freyd, P.J., Scedrov, A.: Categories, allegories. Elsevier (1990)
17. 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)
18. Hinze, R.: Self-certifying railroad diagrams. In: International Conference on Mathematics of Program Construction (MPC). pp. 103–137. Springer (2019)
19. 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)
20. Hyland, M., Schalk, A.: Glueing and orthogonality for models of linear logic. Theoretical Computer Science **294**(1-2), 183–231 (2003)
21. 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)
22. 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)

23. 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)
24. Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. Journal of Pure and Applied Algebra **19**, 193–213 (1980)
25. Kleene, S.C.: Representation of events in nerve nets and finite automata. Tech. rep., RAND PROJECT AIR FORCE SANTA MONICA CA (1951)
26. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2), 366–390 (1994)
27. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems (TOPLAS) **19**(3), 427–443 (1997)
28. Krob, D.: Complete systems of B-rational identities. Theoretical Computer Science **89**(2), 207–343 (1991)
29. Lack, S.: Composing PROPs. Theory and Application of Categories **13**(9), 147–163 (2004)
30. Lambek, J., Scott, P.J.: Introduction to higher-order categorical logic, vol. 7. Cambridge University Press (1988)
31. 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)
32. Moshier, A.M.: Coherence for categories of posets with applications. Topology, Algebra and Categories in Logic (TACL) p. 214 (2015)
33. Piedeleu, R., Zanasi, F.: A string diagrammatic axiomatisation of finite-state automata. arXiv:2009.14576 (2020)
34. Pratt, V.: Dynamic algebras as a well-behaved fragment of relation algebras. In: Proceedings of the International Conference on Algebraic Logic and Universal Algebra in Computer Science. pp. 77–110. Springer (1988)
35. Redko, V.N.: On defining relations for the algebra of regular events. Ukrainskii Matematicheskii Zhurnal **16**, 120–126 (1964)
36. Selinger, P.: A survey of graphical languages for monoidal categories. Springer Lecture Notes in Physics **13**(813), 289–355 (2011)
37. 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)
38. 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)
39. Stefanescu, G.: Network Algebra. Discrete Mathematics and Theoretical Computer Science, Springer London (2000)
40. Thompson, K.: Programming techniques: Regular expression search algorithm. Communications of the ACM **11**(6), 419–422 (1968)
41. Wirth, N.: The programming language pascal. Acta informatica **1**(1), 35–63 (1971)
42. Zanasi, F.: Interacting Hopf Algebras: the theory of linear systems. Ph.D. thesis, Ecole Normale Supérieure de Lyon (2015)