A diagrammatic axiomatisation of finite-state automata

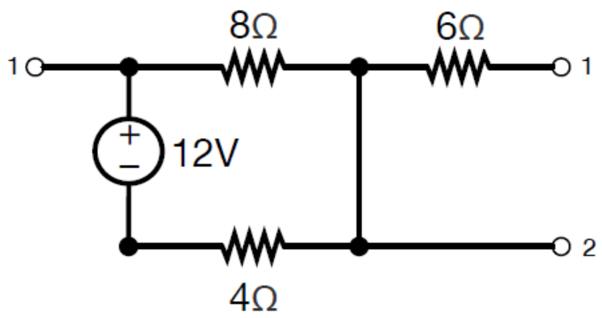
Robin Piedeleu & Fabio Zanasi

arXiv:2009.14576

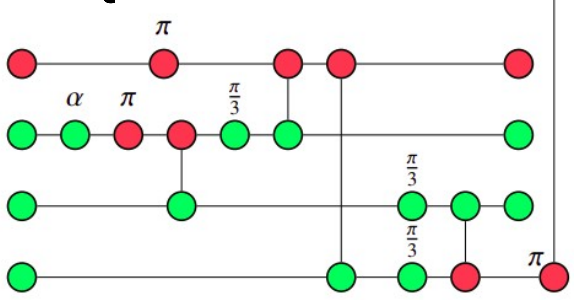
Séminaire PPS, Novembre 2020

String diagrams for open systems

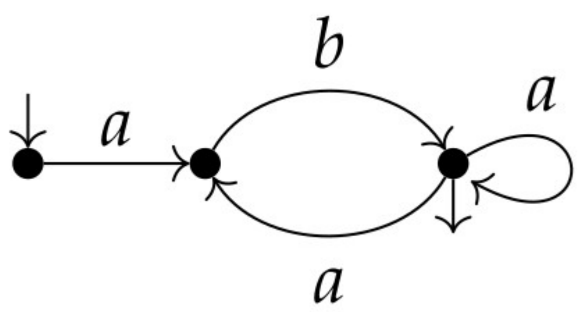
Electrical circuits



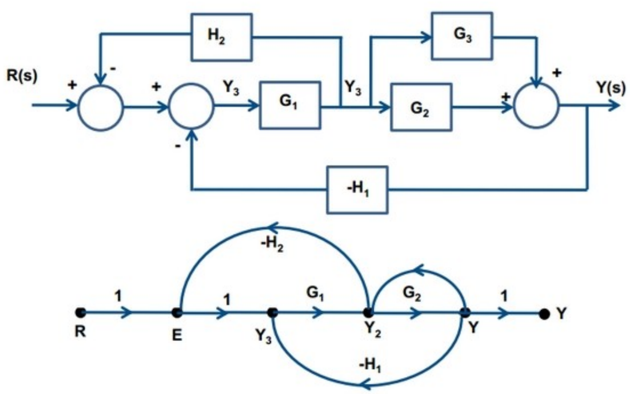
Quantum circuits



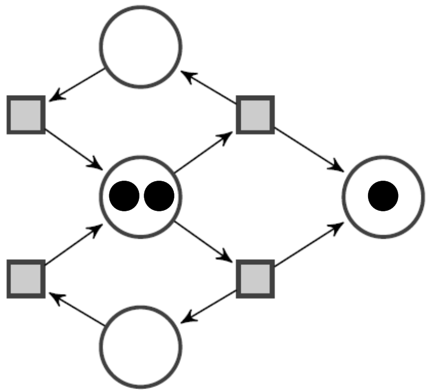
Finite-state automata



Signal flow graphs



Petri nets



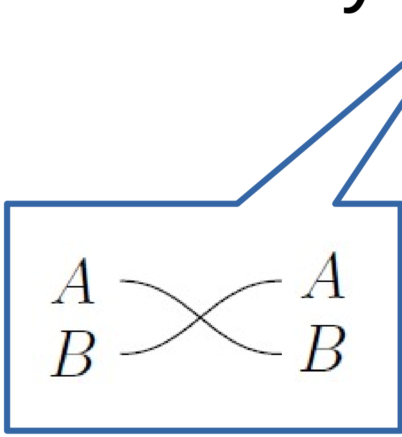
Compositional modelling

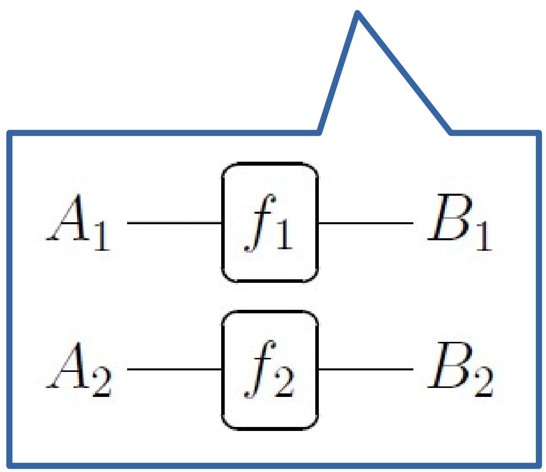
- Compositional = **functorial** semantics:

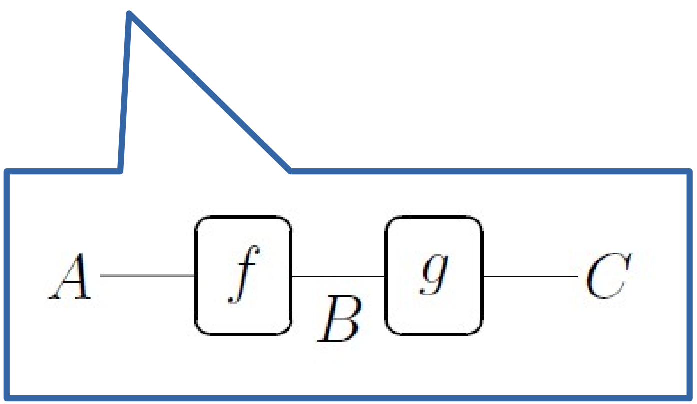
$[\![-]\!]$: 2D Syntax $\longrightarrow$ Behaviour

(String diagrams)

Symmetric monoidal categories







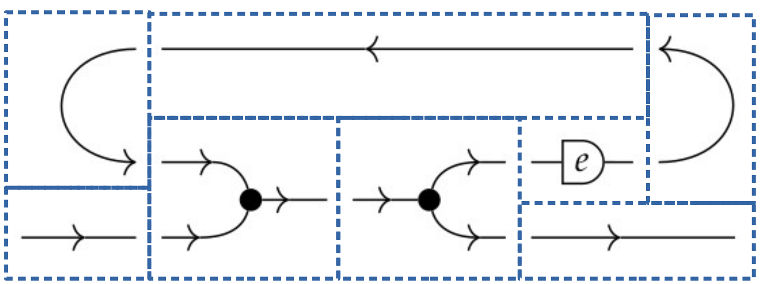
Compositional modelling

- Compositional = **functorial** semantics:

$[\![-]\!] :$ 2D Syntax $\longrightarrow$ Behaviour

(String diagrams)

Symmetric monoidal functor



$$\longrightarrow \{(K,L) \mid \exists M, K \cup eM \subseteq M, M \subseteq L\}$$

The behaviour of the whole can be computed from the behaviour of its parts.

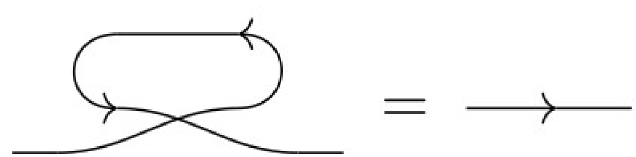
Compositional modelling

- Compositional = **functorial** semantics:

$$[\![-]\!] : \text{2D Syntax (String diagrams)} \longrightarrow \text{Behaviour}$$

Symmetric monoidal functor

- One step further: complete **equational theory**, aka axiomatisation**.**



Today

1. Background
   - Finite-state automata
   - Regular expressions and Kleene algebra
2. Kleene diagrams: first attempt
   - Syntax and Semantics
   - Encoding regexes and NFA
   - Equational theory: the problem with iteration
3. Kleene diagrams: reprise
   - Bringing back regexes
   - Axiomatisation
   - Sketch of the completeness proof
4. Discussion and future work

Background

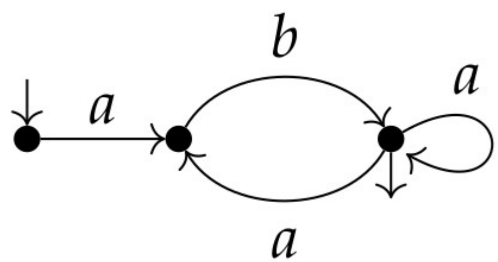
Nondeterministic finite automata

- NFA are traditionally encoded by a tuple

$$(\Sigma, Q, \delta \subseteq Q \times \Sigma \times Q, q_0 \in Q, F \subseteq Q)$$

  (alphabet of basic actions, states, transition relation, initial state, accepting states)
- Example. $\big(\{a, b\}, \{q_0, q_1, q_2\}, \{(q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_1),$ $(q_2, a, q_2)\}, q_0, \{q_2\}\big)$

  more conveniently:



- Recognised language: set of strings $w = a_1 \ldots a_n$ for which there exists a sequence of states $r_0, r_2, \ldots r_n$ such that $r_0 = q_0$, $(r_i, a_{i+1}, r_{i+1}) \in \delta$ *and* $r_n \in F$.

Regular expressions

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

Union

Concatenation

Iteration

Empty set

Empty word

$$[\![e + f]\!]_R = [\![e]\!]_R \cup [\![f]\!]_R \qquad [\![0]\!]_R = \varnothing$$

$$[\![ef]\!]_R = \{vw \mid v \in [\![e]\!]_R, w \in [\![f]\!]_R\} \qquad [\![1]\!]_R = \{\varepsilon\}$$

$$[\![e^*]\!]_R = \bigcup_{n \in \mathbb{N}} [\![e]\!]_R^n \qquad [\![a]\!]_R = \{a\}$$

Kleene theorem. *A language is regular if and only if it is recognised by some NFA.*

Kleene algebra

- Equational presentation of regular expressions:
  - Sum and concatenation (with their units) form an idempotent semiring
  - e^* is the least-fixed point of, e.g., $X = 1 + eX$. But what axioms?
- **Not finitely-based**: no finite set of equations can capture all equalities in the language model [Redko, 1964]
- Finite implicational theory [Kozen, 1994]:

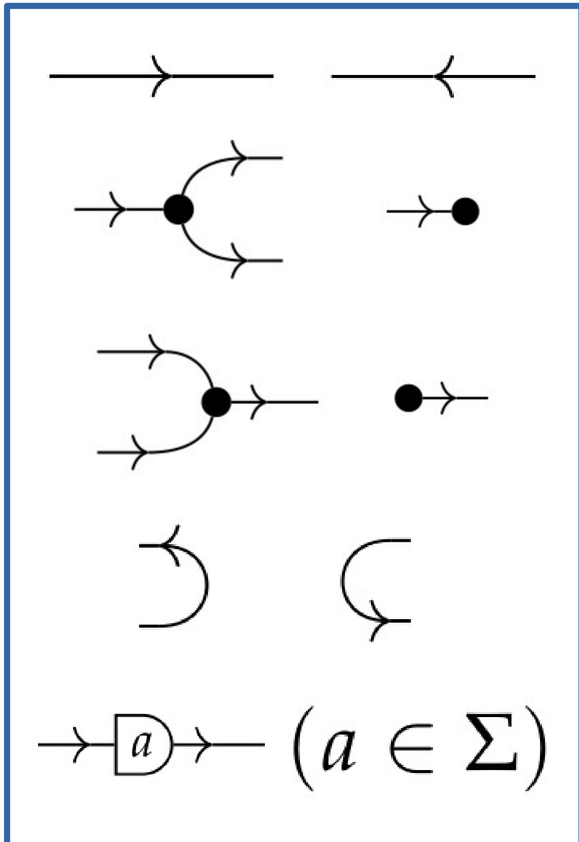
$$1 + ee^* \leq e^* \qquad \text{(star is a fixed-point)}$$

$$f + ex \leq x \Rightarrow e^*f \leq x \qquad \text{(star is the least one)}$$

- Other axiomatisations (some infinitary): Conway, Krob, Salomaa, Kozen, Bloom, Ésik...

Kleene diagrams: first attempt

Diagrams for automata



$\llbracket \cdot \rrbracket$

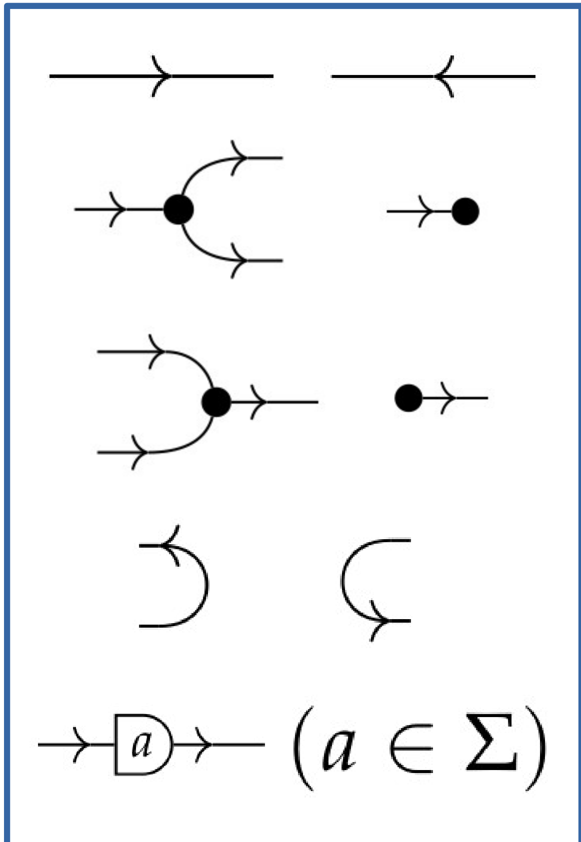
Monotone Relations

(aka Relational/Boolean profunctors,
Weakening relations...)

Objects are posets and morphisms $(X, \leq_X) \to (Y, \leq_Y)$ relations $R \subseteq X \times Y$ such that

$$\textit{if } (x, y) \in R \textit{ and } x' \leq_X x,\ y \leq_Y y' \textit{ then } (x', y') \in R$$

Diagrams for automata



$[\![\cdot]\!]$

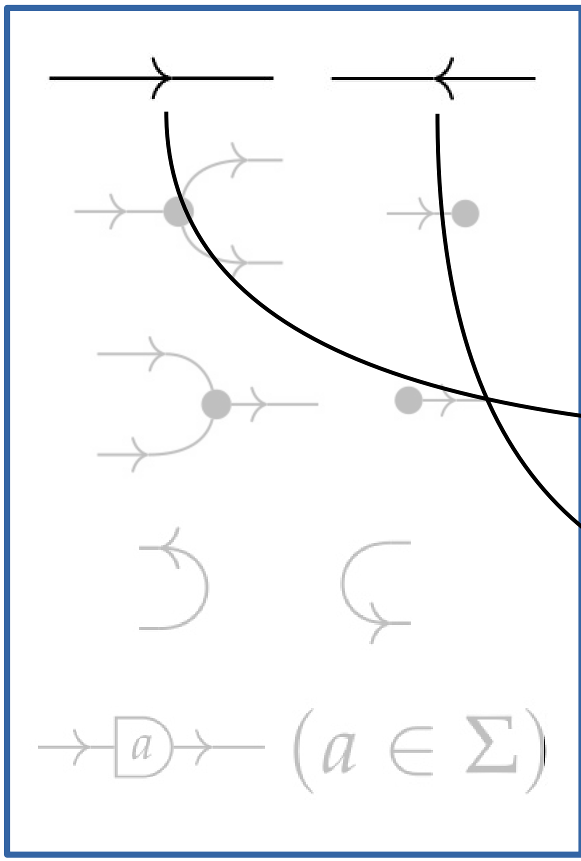
Monotone Relations

(aka Relational/Boolean profunctors,
Weakening relations...)

Compose as relations, with $\leq_X$ as identity $(X, \leq_X) \to (X, \leq_X)$.

Symmetric monoidal category with product of posets.

Diagrams for automata



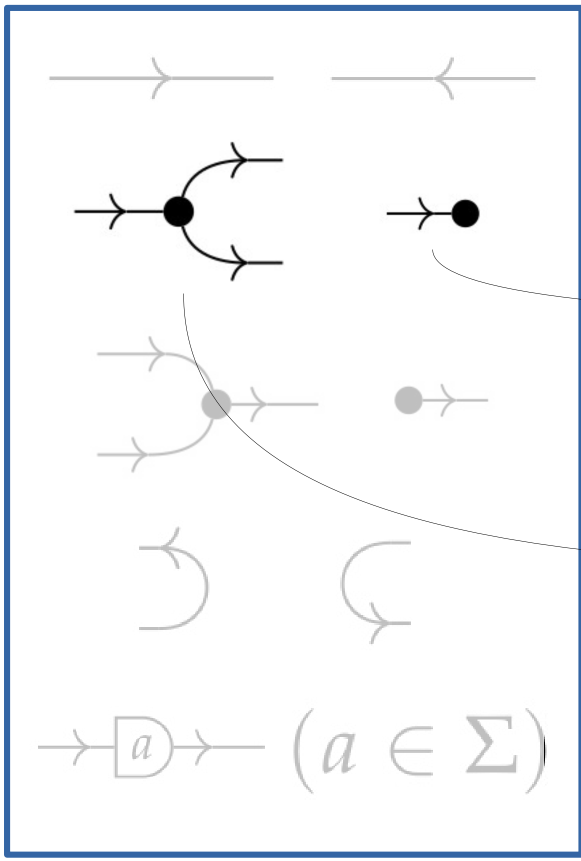
$[\![\cdot]\!]$

Two generating objects with identities given by the inclusion relations on languages:

$$\{((L,K), L \subseteq K) \mid L,K \subseteq \Sigma^\star\}$$

$$\{((L,K), K \subseteq L) \mid L,K \subseteq \Sigma^\star\}$$

Diagrams for automata



$[\![\cdot]\!]$

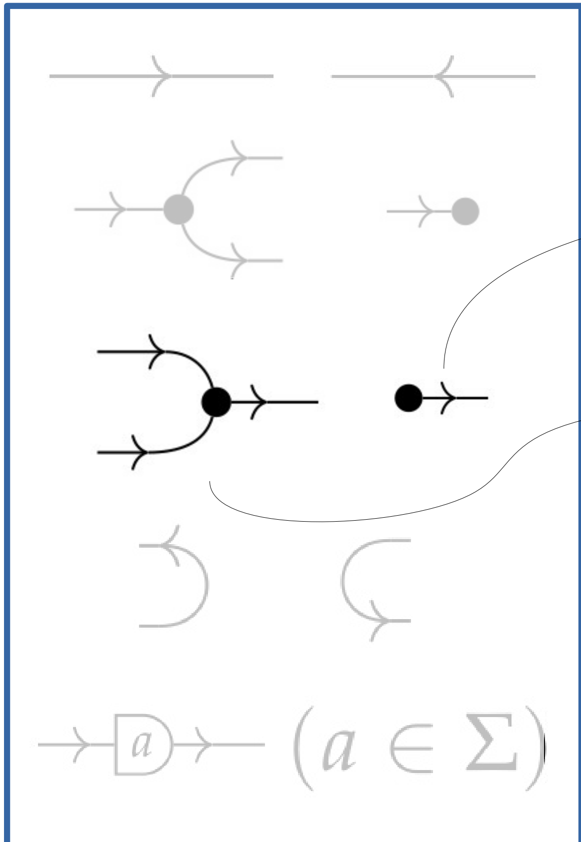
Delete

$$\{(L, \bullet) \mid L \subseteq \Sigma^\star\}$$

Copy

$$\Big\{\big(L, (K_1, K_2)\big) \mid L \subseteq K_i,\ i = 1, 2 \quad L, K_1, K_2 \subseteq \Sigma^\star\Big\}$$

Diagrams for automata



$[\![\cdot]\!]$

Empty set

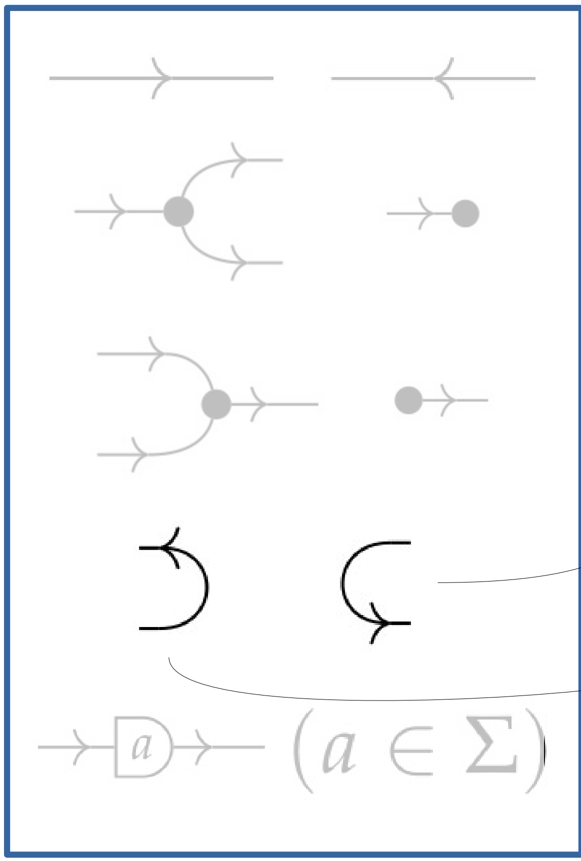
$$\{(\bullet, L) \mid L \subseteq \Sigma^\star\}$$

Union

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

$$= \{((L_1, L_2), K) \mid L_1 \cup L_2 \subseteq K,\ L_1, L_2, K \subseteq \Sigma^\star\}$$

Diagrams for automata



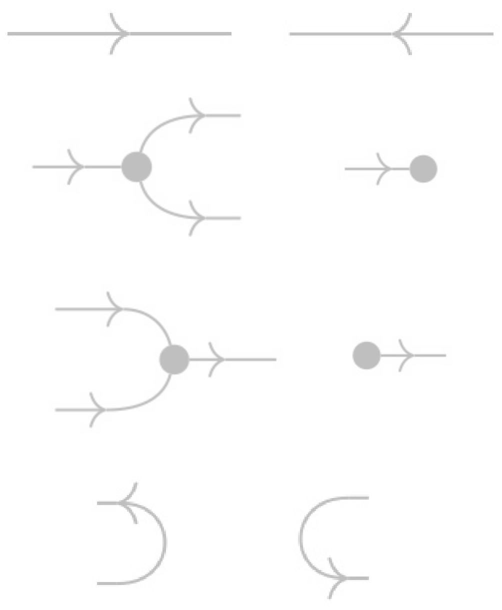
$[\![\cdot]\!]$

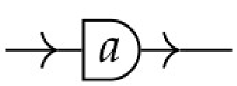
Plumbing

$$\{(\bullet, (L, K)) \mid L \subseteq K,\ L, K \subseteq \Sigma^\star\}$$

$$\{((L, K), \bullet) \mid K \subseteq L,\ L, K \subseteq \Sigma^\star\}$$

Diagrams for automata





$(a \in \Sigma)$

$\xrightarrow{[\![\cdot]\!]}$

Right-action of Σ *by concatenation*

$$\{(L, K) \mid La \subseteq K,\ L, K \subseteq \Sigma^\star\}$$

Compositionality

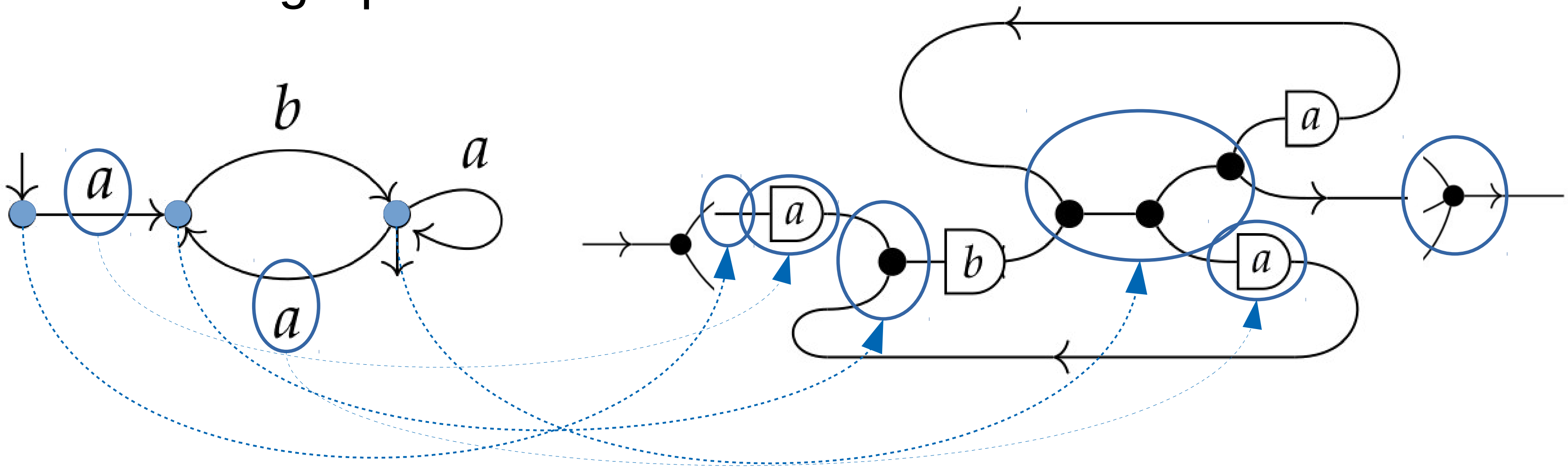
...means that

$$\left\llbracket -\boxed{c}-\boxed{d}- \right\rrbracket = \left\{ (L,K) \mid \exists M\, (L,M) \in \left\llbracket -\boxed{c}- \right\rrbracket, \right.$$
$$\left. (M,K) \in \left\llbracket -\boxed{d}- \right\rrbracket \right\}$$

$$\left\llbracket \begin{matrix} -\boxed{c_1}- \\ -\boxed{c_2}- \end{matrix} \right\rrbracket = \Big\{ \big((L_1, L_2), (K_1, K_2) \big)$$
$$\mid (L_i, K_i) \in \left\llbracket -\boxed{c_i}- \right\rrbracket, \, i = 1,2 \Big\}$$

Sanity check: NFA

- Formal encoding from tuples definition is tedious.
- Intuition via graphical notation:

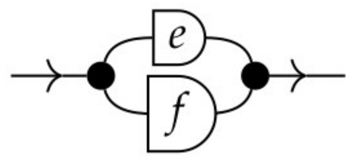


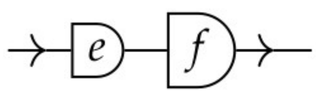
- **Theorem.** Given an NFA which recognises a language *L*, the semantics of its associated diagram, constructed as above, is

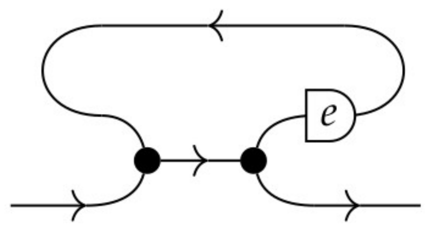
$$\{(K, K') \mid KL \subseteq K'\}$$

Sanity check: regexes

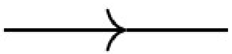
- We can encode regexes as follows

$\langle e + f \rangle =$ 

$\langle ef \rangle =$ 

$\langle e^* \rangle =$ 

$\langle 0 \rangle =$ 

$\langle 1 \rangle =$ 

$\langle a \rangle =$ 

- **Proposition**. The encoding preserves the semantics, i.e., for any expression *e*,

$$[\![\langle e \rangle]\!] = \{(L, K) \mid L\, [\![e]\!]_R \subseteq K\}$$

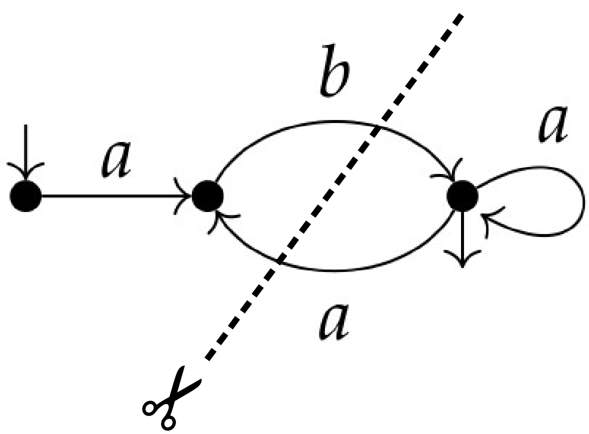
Semantic functor

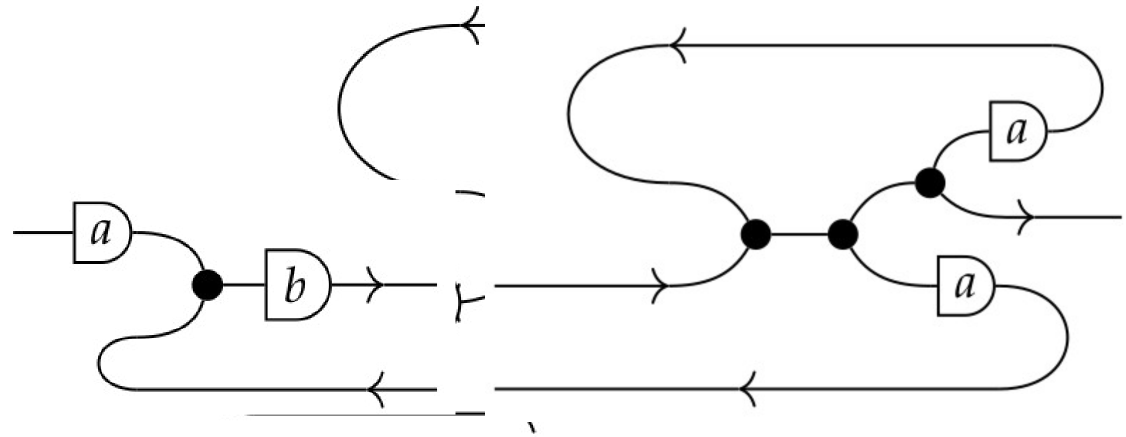
Regex encoding

Standard regex interpretation

What else?

- Benefits of (de)compositionality



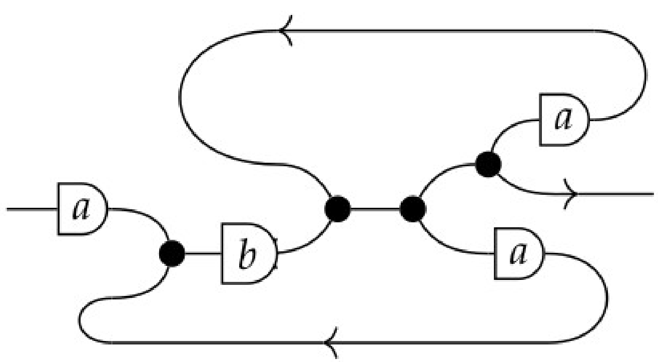


- Gives formal status to automata with **multiple inputs/outputs**.
- But **nc** **e expressive**: every diagram is fully of

Beware! Do not necessarily coincide with initial/accepting states in the usual definition.

A more concrete view

A diagrammatic language to specify systems of **linear** language inequalities, i.e. for which concatenation is restricted to left-action of letters.

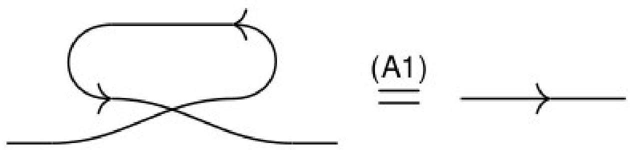


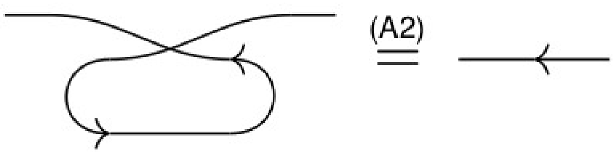
$$\begin{cases} \epsilon \subseteq X_0 \\ X_0 a \subseteq X_1 \\ X_1 b \subseteq X_2 \\ X_2 a \subseteq X_1 \\ X_2 a \subseteq X_2 \end{cases}$$

Equational theory

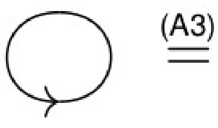
Plain wires

- We have a **compact closed** category: we can bend/straighten wires at will, keeping track of only their orientation



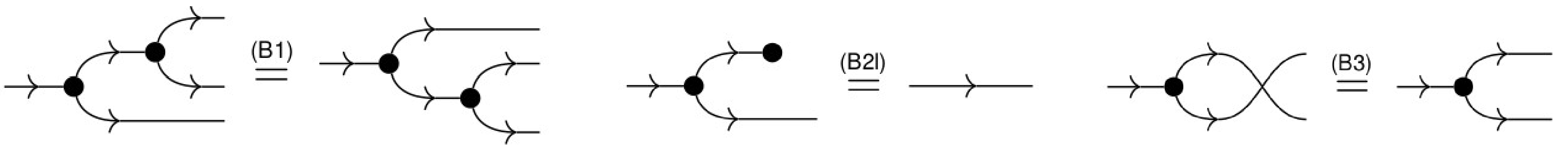


- … and we can eliminate isolated loops

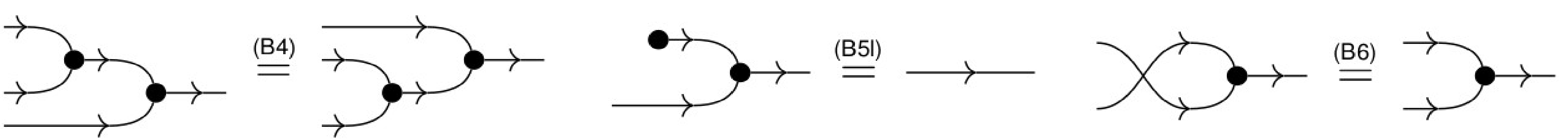


Copy and Sum

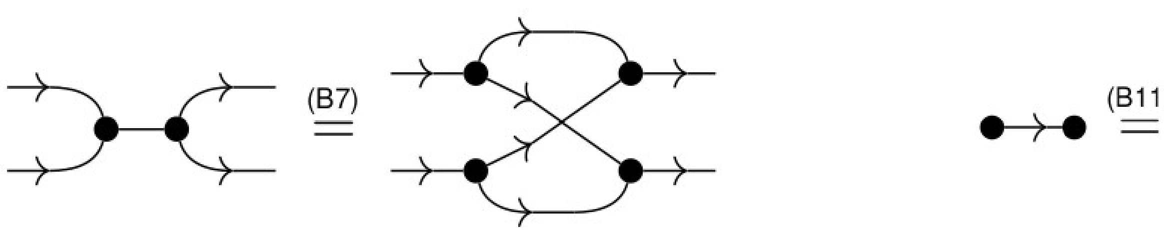
- Cocommutative comonoid



- Commutative monoid

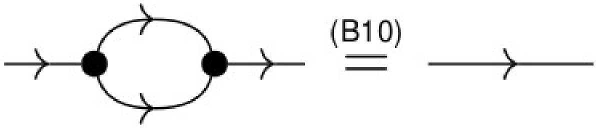


- Bimonoid

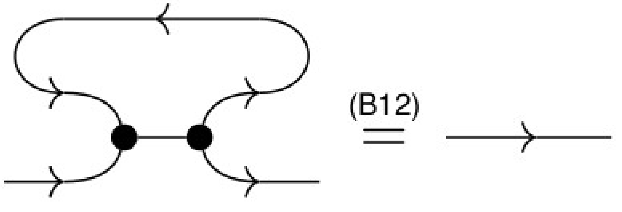


Copy and Sum

- Idempotent

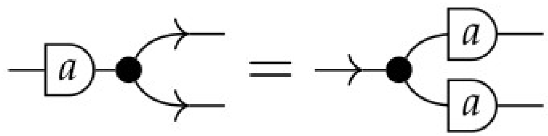


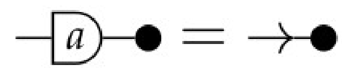
- Getting rid of trivial feedback



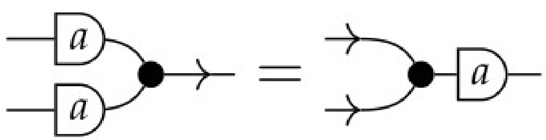
Concatenation

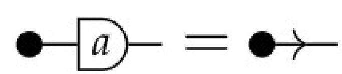
- Letters can be copied and deleted...





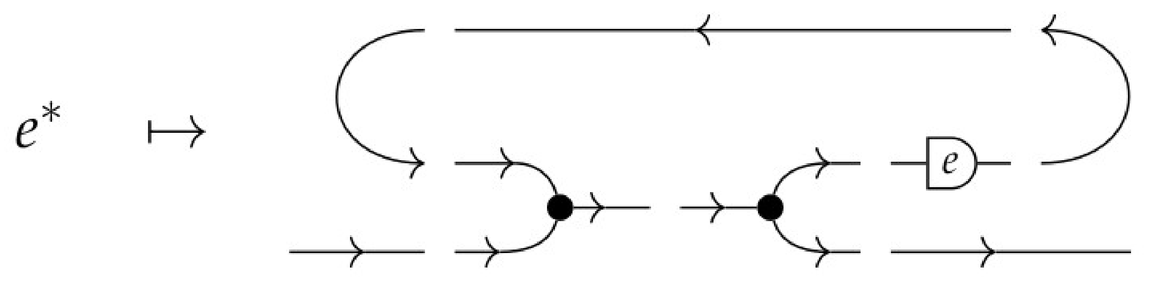
- ...merged and spawned





The problem with iteration

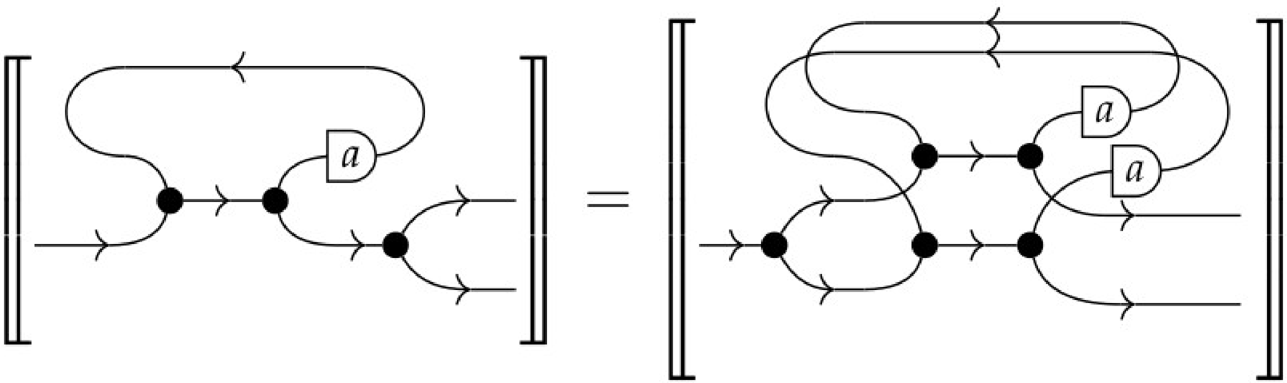
- Recall: Kleene algebra **not finitely-based** in the standard algebraic setting. The main obstacle is iteration (represented by the star).
- Here it is a **derived** notion, made up of more primitive components:



- But the problem did not disappear.

The problem with iteration

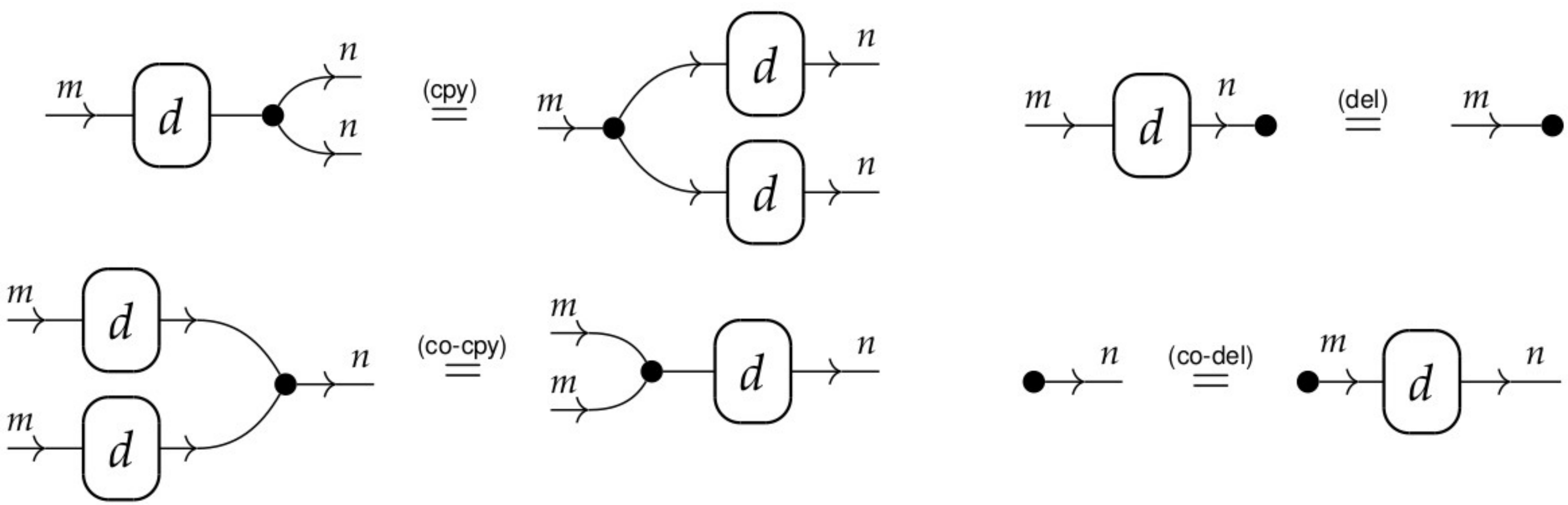
- Simple check: we should be able to copy/delete/merge/spawn an expression in a loop. For example,



- Incompleteness: we cannot prove this with just the current axioms.
- Even if we add it, we need to be deal with **arbitrary nestings** of loops with other operations.

One solution

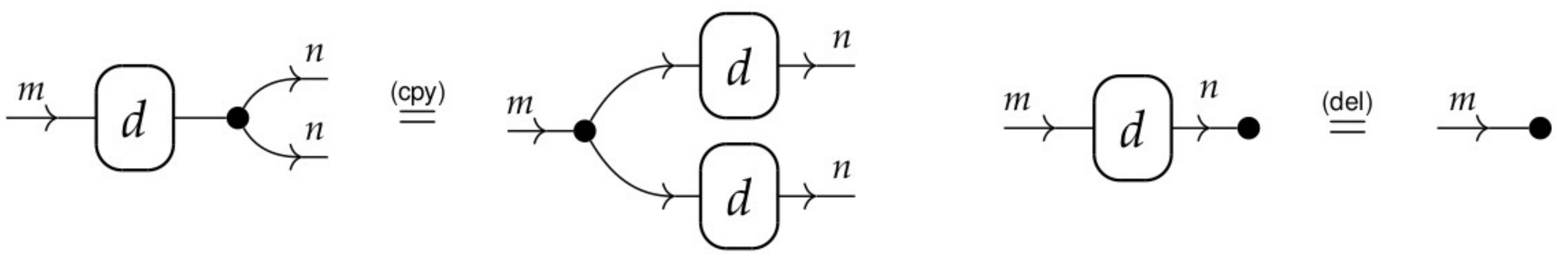
- Impose global (so infinitary) **axiom schemes**.
- **Definition.** A diagram is *left-to-right* if it has all inputs in its domain and all outputs in codomain.
- For any left-to-right diagram *d*, we want



- By *fiat*: similar to **matricial iteration theories** [Bloom and Ésik, 93] although, even relative to this setting, they did not produce a finitary axiomatisation for regular languages.

Semantics of least fixed-points

- Monotone maps embed into monotone relations: *f* is sent to $\{(x,y) \mid f(x) \le y\}$.
- A relation satisfies copying and deleting,

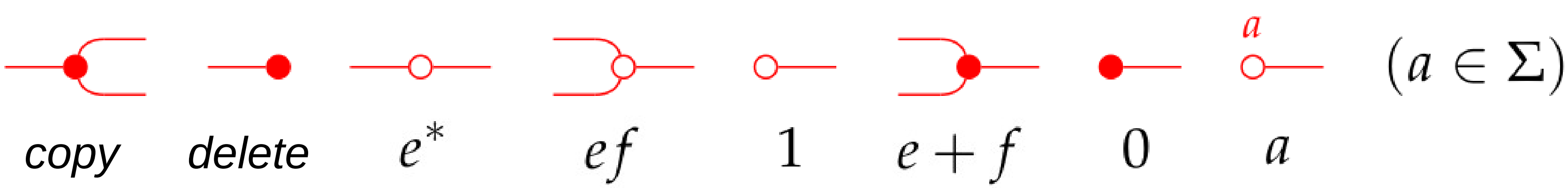


  iff it is the image of a monotone **map**.
- The semantics of e^* is the least fixed-point of the language map $f = \lambda Z.\ X \cup eZ$. This is still (the image of) a monotone map in X, *i.e.*,
  - (del) means the least fixed-point **exists** for every X;
  - (cpy) means it is **unique**.

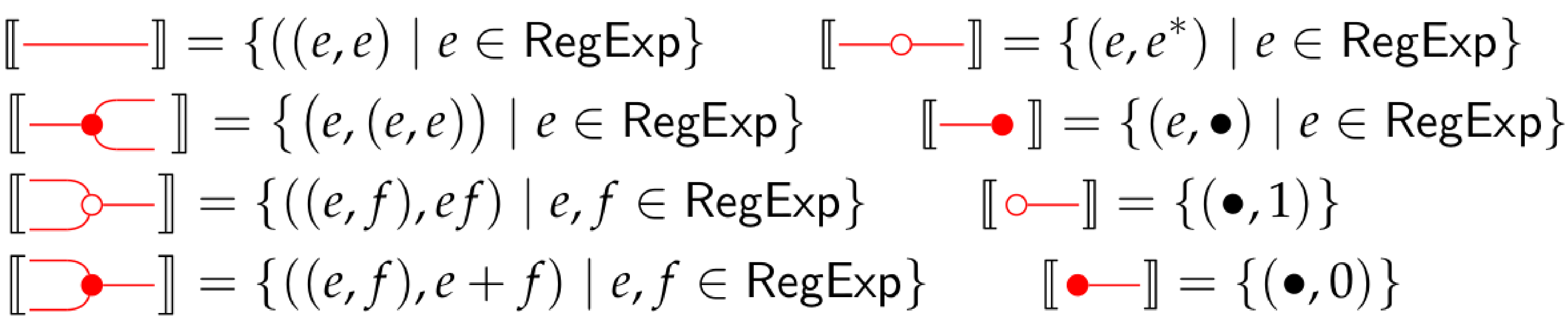
Kleene diagrams: reprise

A trick: bringing back regexes

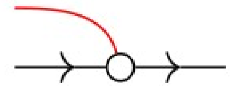
- Extend the syntax with **regular expressions** on a separate wire type:

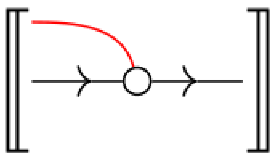


- Note that this is **just syntax**. Their interpretation is the free term algebra of regexes.



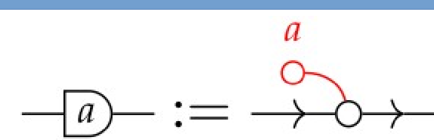
A trick: bringing back regexes

- Syntax: replace  with general action of **any regex** (not just the letters) via 
- Semantics: regex **acting on languages** by concatenation on the left

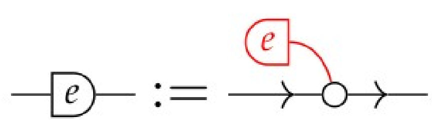
 $= \{((e, L), K) \mid L\,[\![e]\!]_R \subseteq K \text{ and } e \in \mathsf{RegExp}, L, K \subseteq \Sigma^\star\}$

Interpretation of the regex *e* (a regular language)

Free (uninterpreted) term algebra of regexes

- We recover the atomic actions as 

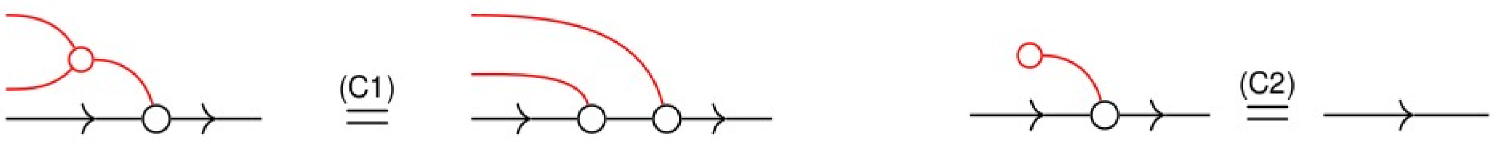
- String diagrams for **generalised automata** with transitions labelled by arbitrary regexes:



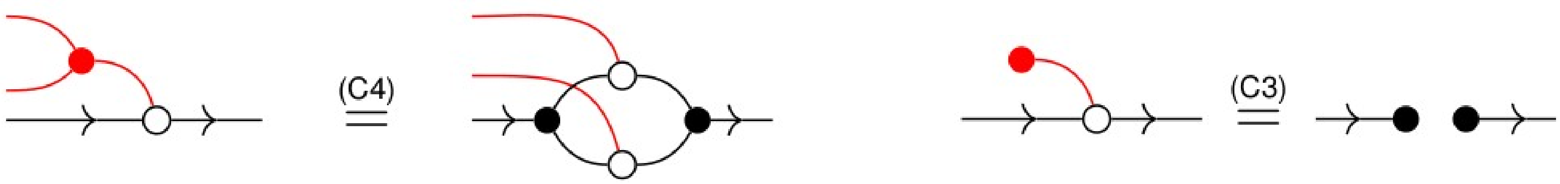
Axiomatising the action (1/2)

Capturing the behaviour of the action:

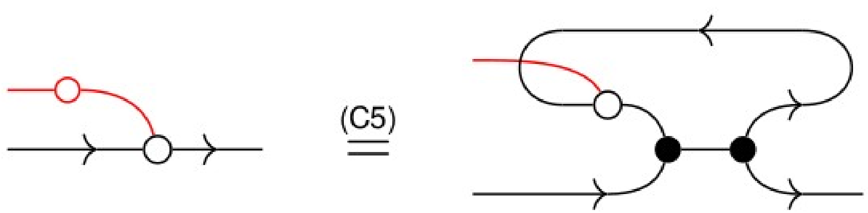
- Concatenation and empty word



- Union and empty language

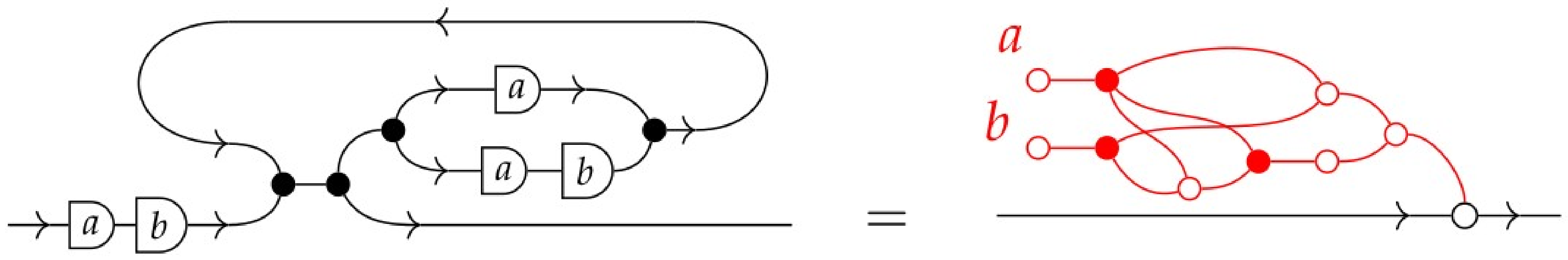


- Iteration



Unfolding/compiling regexes

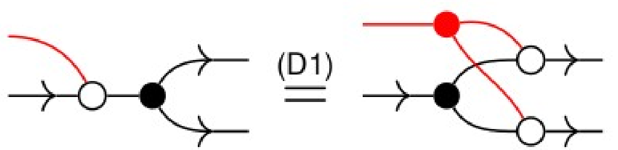
Example. $ab(a+ab)^*$

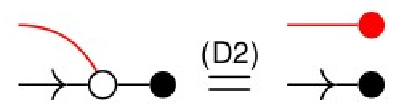


Axiomatising the action (2/2)

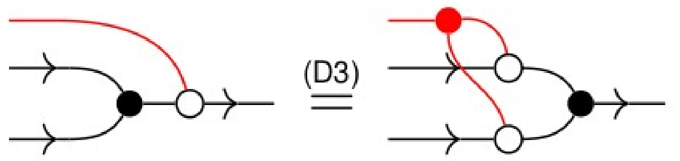
Back to the original problem:

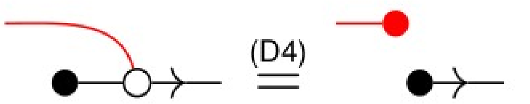
- Copy and delete arbitrary regexes





- Merge and spawn arbitrary regexes





Theorem (Completeness). *Two diagrams are equal iff they are mapped to the same monotone relation.*

Completeness proof outline

- Normal form argument: diagrammatic counterpart of constructing the **minimal deterministic** automaton that recognises the same language
  - An automaton is deterministic (DFA) if its transition relation is the graph of a function $Q \times \Sigma \to Q$.
  - Among the finite-state automata that recognise a given language, there is a **unique** DFA with the **smallest number of states**. This is our normal form.
- Obtained via **Brzozowski's algorithm**, implemented as equational reasoning:

$$\underbrace{\text{reverse; determinise; reverse;}}_{\text{Just determinisation in reverse: immediate by the symmetries of the equational theory.}}\ \underbrace{\text{determinise}}_{\text{Key step}}$$

Completeness proof outline

- Normal form argument: diagrammatic counterpart of constructing the **minimal deterministic** automaton that recognises the same language
  - An automaton is deterministic (DFA) if its transition relation is the graph of a function $Q \times \Sigma \to Q$.
  - Among the finite-state automata that recognise a given language, there is a **unique** DFA with the **smallest number of states**. This is our normal form.
- Obtained via **Brzozowski's algorithm**, implemented as equational reasoning:

$$\underbrace{\text{reverse; determinise; reverse;}}_{\substack{\text{Just determinisation in reverse: immediate}\\ \text{by the symmetries of the equational theory.}}}\ \underbrace{\text{determinise}}_{\text{Key step}}$$

Determinisation, traditionally

For an NFA given by the tuple $(\Sigma, Q, \delta, q_0, F)$

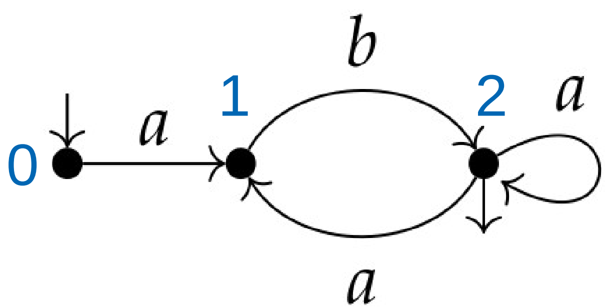
an equivalent (i.e. that recognises the same language) DFA is given by

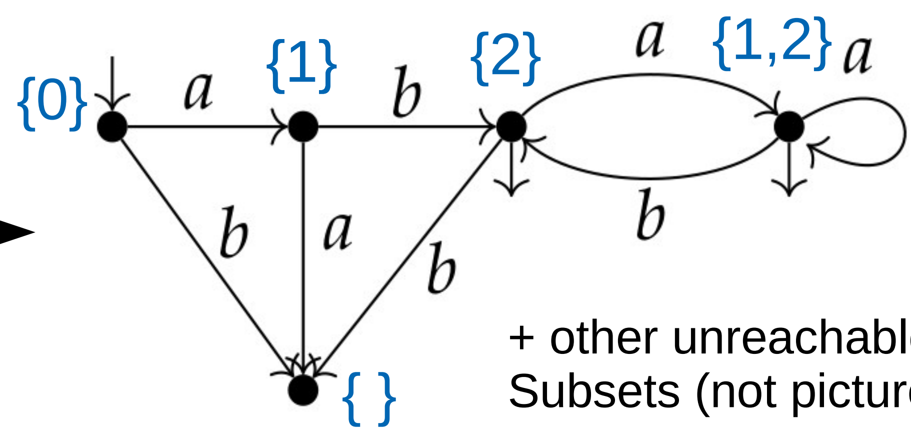
$$(\Sigma, 2^Q, \Delta, \{q_0\}, G)$$

where

$$(S, a, S') \in \Delta \text{ iff } S' = \{q' \in Q \mid \exists q \in S, (q, a, q') \in \delta\}$$

and *G* is the set of subsets of *Q* that contain at least one accepting state.

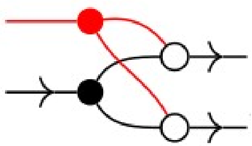
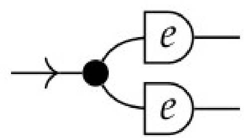
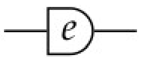
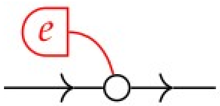




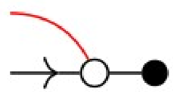
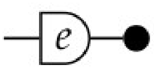
+ other unreachable Subsets (not pictured)

Determinisation, diagrammatically

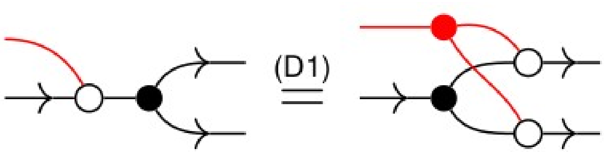
- **Nondeterministic** transitions of automata correspond to subdiagrams of the form

 (or  where  := )

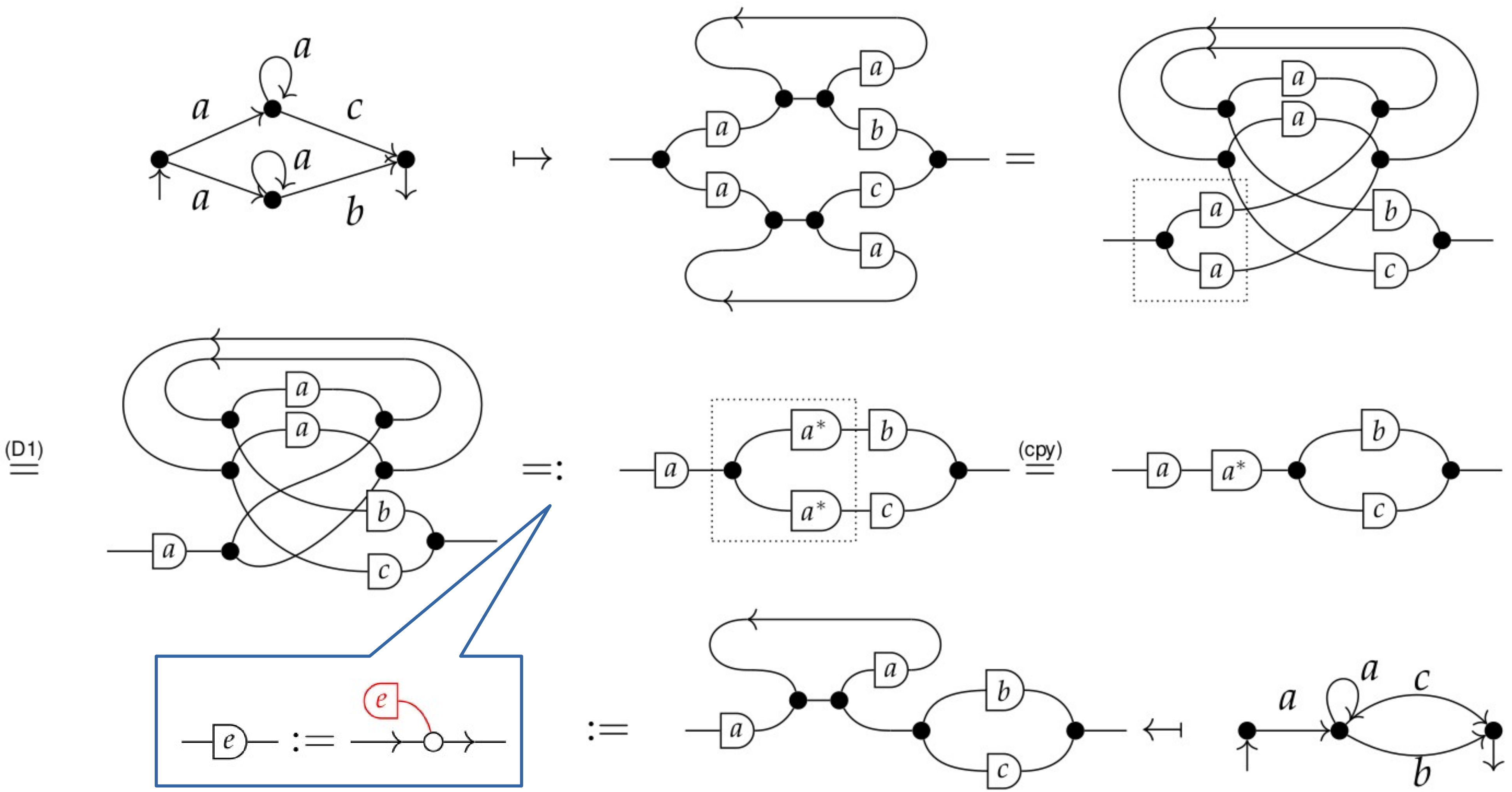
- **Useless** states (those that cannot reach an accepting state/ contribute to the semantics) correspond to subdiagrams of the form

 (or )

- To get rid of them, just apply (not haphazardly, check the paper for details):

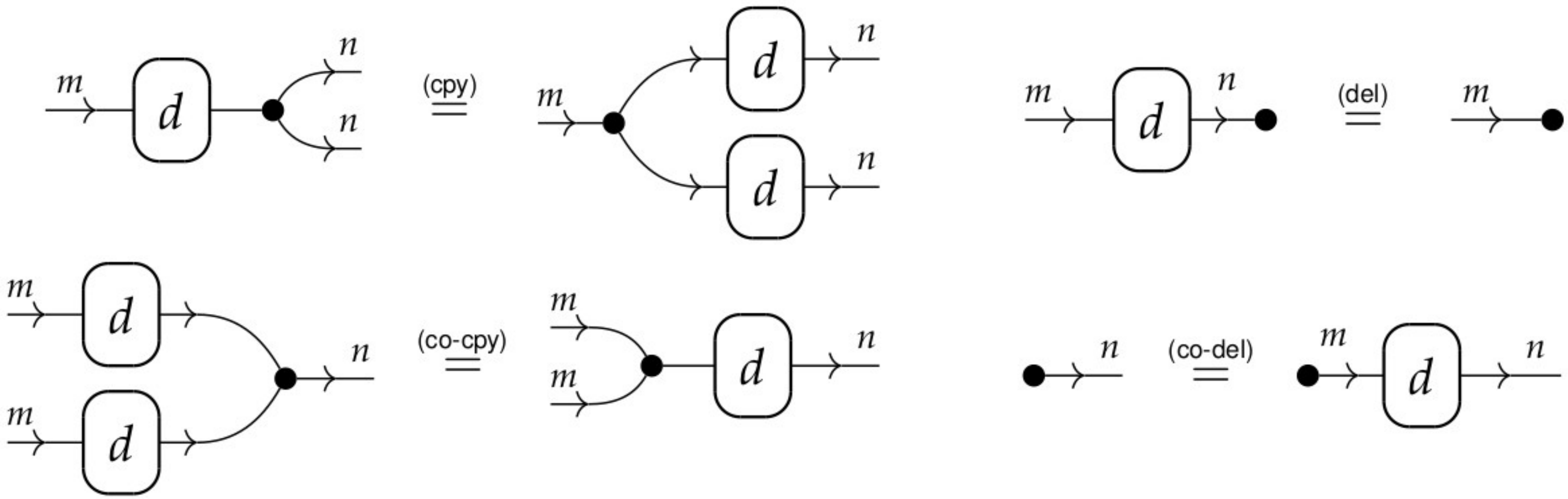


Diagrammatic determinisation example



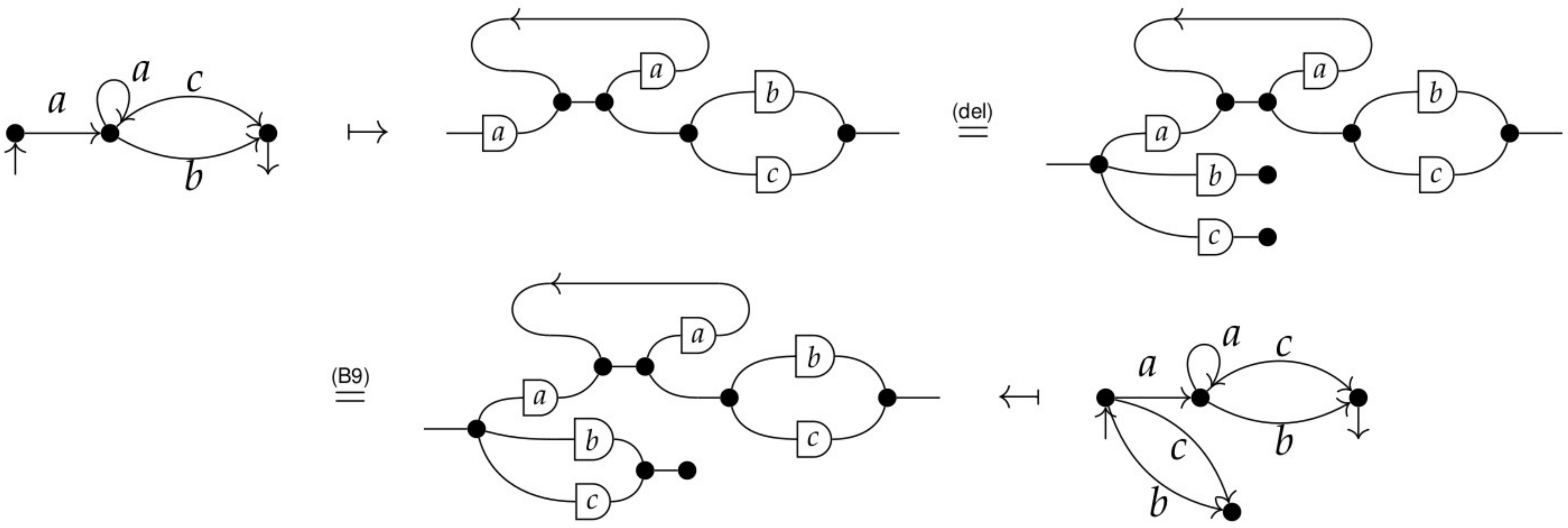
Left-to-right diagrams again

- Now we can prove that, for any left-to-right diagram *d*

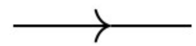
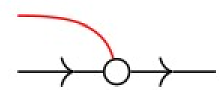
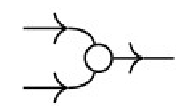


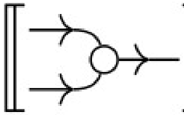
- *Subcategory of left-to-right diagrams maps to a* ***category of matrices over the semiring of regular languages****, with matrix product as composition and direct sum as product.*
- *Two uses: 1) reduces the completeness proof to diagrams with one input and one output; 2) is the engine of the diagrammatic determinisation procedure.*

Diagrammatic determinisation example

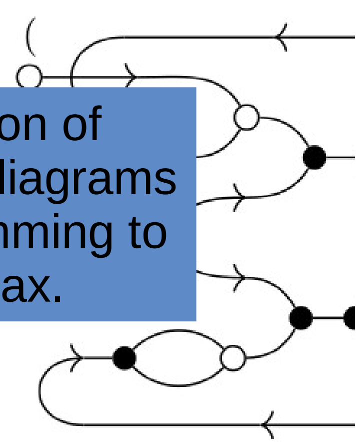


Bonus: context-free languages

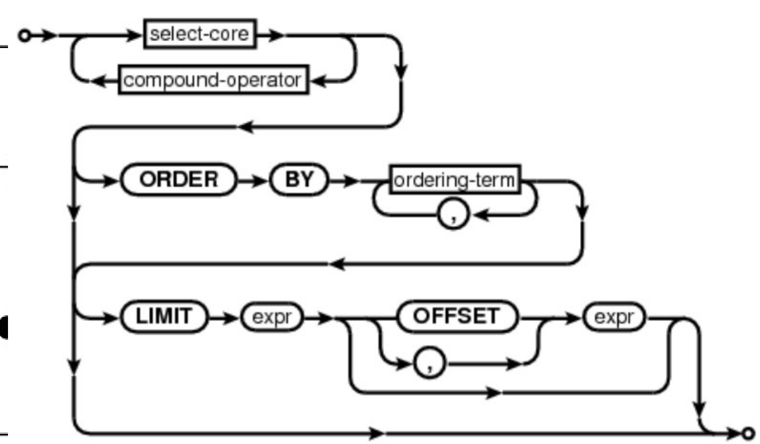
- Recall that we designed a language to specify systems of **linear** language inequations.
- Remove the linearity constraint: unconstrained concatenation gives systems of polynomial language inequations.
- Diagrammatically, turn into  and  into  with

[[]] $= \{((M, L), K) \mid ML \subseteq K\}$

- We can specify context-free languages. For example, the language of properly matched parentheses:



Formal version of syntax/railroad diagrams used in programming to define syntax.



Discussion

- **What's new?** A finite presentation of a symmetric monoidal category (SMC) that axiomatises automata equivalence.
- **In what sense is it finite?** Debatable: not in the usual sense of algebraic theories, but relative to the equational theory of SMCs.
  - If we encode terms using only ; and $\otimes$, it is infinite.
  - But we should encode them as graphs (and equations as graph rewrites).
- **Is it really new?** All previous work was in a traced symmetric monoidal setting (*iteration theories* of Bloom & Ésik or *network algebra* of Stefanescu). But:
  - Still no finite axiomatisations of regular languages in these settings.
  - The trace is a global operation that cannot be finitely axiomatised relative to the theory of symmetric monoidal categories.

Discussion

- **Why does this work?** Slightly mysterious, perhaps better compositionality.
  - Not the first time that a finite axiomatisation of a theory that is provably not finitely-based in the standard algebraic setting: *graphical conjunctive queries* vs. *allegories*, for example.
  - Proofs of negative results in the algebraic setting rely on showing the correspondence between terms and certain *graphs*. The two-dimensional syntax allows to represent all graphs/automata.

Future work

- This category of monotone relations over languages has very rich hidden structure:
  - Cartesian **bicategory** with inclusions of relations as 2-cells;
  - **adjoints** (in the bicategorical sense) **to copy and sum** that represent the Boolean lattice structure of languages (not just the order).
- Using this more expressive language, a generalisation of **bisimulation** can be defined and is sufficient to prove that two diagrams corresponding to equivalent automata are equal.
- But **not yet a complete equational theory** for the whole extended syntax.
- This seems to correspond to **alternating automata**.

Questions?