Submitted to MFPS 22

A Complete Diagrammatic Calculus for Boolean Satisfiability

Tao Gu Robin Piedeleu Fabio Zanasi

University College London

Abstract

We propose a calculus of string diagrams to reason about satisfiability of Boolean formulas, and prove it to be sound and complete. We then showcase our calculus in a few case studies. First, we consider SAT-solving. Second, we consider Horn clauses, which leads us to a new decision method for propositional logic programs equivalence under Herbrand model semantics.

Keywords: String diagrams, Categorical semantics, Boolean algebra, Satisfiability, Logic programming

1 Introduction

In the 1970s, Cook and Levin proved independently [13,22] that a number of difficult problems reduce to that of determining the existence of a satisfying assignment to a given Boolean formula. For this reason, in spite of its (suspected) theoretical intractability, and perhaps because of its surprising practical feasibility, the problem of Boolean satisfiability has acquired a central importance in logic and computer science.

This paper proposes a new perspective on the algebra of Boolean satisfiability, in the form of a sound and complete calculus of string diagrams.

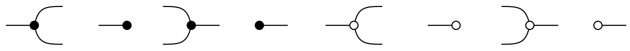
At first glance, it may appear we already have a satisfactory algebraic treatment of Boolean satisfiability, dating back to the landmark works of George Boole in the 19th century [8]. Thanks to Boole, we know that classical propositional logic can be formulated as an *algebraic theory*, whose models are what we now call Boolean algebras. However, this standard treatment has a catch, which is the starting point of our work: *the satisfiability of a propositional formula cannot be stated as a sentence in the algebraic theory of Boolean algebras.* The reason is simple – algebraic theories only allow axioms as equations between terms with free variables (or, equivalently from the point of view of first-order classical provability, as equations in which every variable is universally quantified). On the other hand, a Boolean formula f containing free variables $x_1, \ldots, x_n$, is satisfiable if and only if $\exists x_1 \ldots \exists x_n(f = 1)$, which is outside of the algebraic realm. Call this formula SAT(f); the algebraic theory of Boolean algebras is insufficiently expressive to encode SAT(f) as a statement in the theory itself.

Of course, it is a perfectly well-formed first-order statement, so we could just use first-order logic to reason about SAT and derive the (un)satisfiability of particular instances. However, we argue that a genuinely algebraic treatment of SAT is possible and preferable, provided that we move to a *diagrammatic* syntax. To this effect, we will introduce a calculus to compose systems of Boolean constraints expressed as string diagrams, and reason about them in a purely equational way.

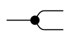
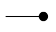
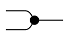
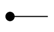
[1] Thanks to Guillaume Boisseau and Cole Comfort for helpful discussion on the completeness proof and related work.

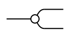
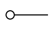
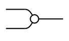
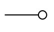
Why would we need a *diagrammatic* syntax? As we have just shown, the algebraic theory of Boolean algebras is insufficiently expressive, thus we are looking for some other formal system that is more closely tailored to satisfiability. Existential quantification gives the problem a particularly relational flavour: SAT(f) does not involve evaluating a Boolean function, but checking that there exists some assignment for which the function f evaluates to true. This is a fundamentally *relational*, not functional constraint. And diagrammatic calculi are particularly well suited to express relational constraints, as demonstrated by the wealth of diagrammatic languages developed in recent years for different subcategories of relations (linear [29,7], polyhedral [3], affine [5], piecewise-linear [2]). In this setting, string diagrams have the advantage of highlighting key structural features, such as dependencies and connectivity between different sub-terms/diagrams. Finally, another related way to think about string diagrams is as a multi-sorted algebraic syntax, generalising standard algebraic syntax to the regular fragment of first-order logic, *i.e.* the fragment containing truth, conjunction, and existential quantification [6,4].

In order to develop our approach, we take as semantic building blocks certain sets of satisfying assignments of Boolean formulas, which we organise into a symmetric monoidal category. We then devise a diagrammatic syntax for them generated by

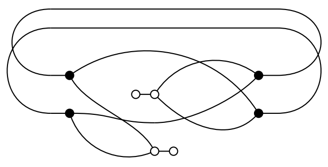


A diagram in this syntax is formed by composing any of the generators horizontally or vertically, connecting and crossing wires where needed. A dangling wire represents a free variable so that a diagram with m wires on the left and n wires on the right can be thought of as a Boolean formula in CNF with $m + n$ free variables. The intuition is that variables corresponding to wires on the left appear negatively in the associated formula, while those on the right appear positively. The correspondence between the syntax and the Boolean semantics is simple: each diagram is interpreted as the set of satisfying assignments of an associated formula. It is helpful to give these formulas in CNF for the generators explicitly:

 $\mapsto (\neg x \vee y_1) \wedge (\neg x \vee y_2)$  $\mapsto 1$  $\mapsto (\neg x_1 \vee y) \wedge (\neg x_2 \vee y)$  $\mapsto 1$

 $\mapsto \neg x \vee y_1 \vee y_2$  $\mapsto y$  $\mapsto \neg x_1 \vee \neg x_2 \vee y$  $\mapsto \neg x$

with the convention that xs correspond to left wires and ys to right wires. Then, composing two diagrams in parallel (vertically) amounts to taking the conjunction of their associated formulas, while composing them in series (horizontally) requires existentially quantifying over the variable corresponding to the shared wire, effectively projecting it out. Using these, we are able to encode arbitrary SAT instances – statements of the form $\exists x f$ for some CNF formula f – as diagrams without any dangling wires. For example, the SAT instance corresponding to $\exists x \exists y (\neg x \vee y) \wedge (x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$ can be depicted as on the right.



The main technical result of this paper is a sound and complete equational theory for the intended semantics: any (in)equality that holds in the Boolean semantics, can be derived by equational reasoning at the level of the diagrams themselves. This theory, we argue, is the *algebra of satisfiability*.

Using our complete axiomatisation, we can derive the (un)satisfiability of any given SAT instance entirely diagrammatically, by applying local equational reasoning steps. For example, in a few steps, diagram above can be rewritten to $\circ\!\!-\!\!\circ$, a diagram which represents the unsatisfiable formula $\exists x (\neg x \wedge x)$.

We also take advantage of the fact that our syntax can represent *Horn clauses* to give a sound and complete encoding of propositional logic programs. We show how to represent definite logic programs diagrammatically and how their conventional semantics – their *least Herbrand model* – corresponds to the semantics of the associated diagram. Paired to the completeness result, this gives us a procedure to decide semantic equivalence of logic programs, by rewriting in the equational theory of our calculus — see e.g. Example 5.8 below.

Synopsis. We introduce and motivate the semantic domain of interest in Section 2. We introduce the diagrammatic calculus in Section 3. We show how to use the calculus to reason about satisfiability in Section 4, and how to use it as a proof system for equivalence of logic programs in Section 5. We conclude with the proof of completeness in Section 6.

2 Boolean formulas, Satisfiability and Monotone Relations

In this section we review the basics of Boolean formulas, and justify the introduction of monotone relations as a means to reason about their satisfiability.

We write $\mathbb{B} = \{0, 1\}$ for the two-element Boolean algebra, with the standard order $\{(0, 0), (0, 1), (1, 1)\}$, and $\mathbb{B}^{op}$ for the same set with the opposite order. A *Boolean formula* is either 0, 1, a variable (from some given set of variables $x, y, \dots$), a conjunction of Boolean formulas $f_1 \wedge f_2$, a disjunction of Boolean formulas $f_1 \vee f_2$, or the negation of a Boolean formula $\neg f$. Any Boolean formula with n free variables admits a standard interpretation as a map $\mathbb{B}^n \to \mathbb{B}$ given by the usual Boolean semantics of conjunction, disjunction and negation (via their truth-tables). As is common practice, we will use the same notation to denote a Boolean formula f and its interpretation $f \colon \mathbb{B}^n \to \mathbb{B}$.

We call *literal* a variable or the negation of a variable. We say that a formula is in *conjunctive normal form* (also known as clausal form) (CNF) when it is given as a conjunction of disjunctions of literals, *e.g.* $(\neg x \vee y \vee z) \wedge (x \vee z \vee \neg y)$. Each of the disjunctions of literals is called a clause. De Morgan duality guarantees that every Boolean formula admits a CNF formula with the same interpretation, *i.e.* that defines the same map $\mathbb{B}^n \to \mathbb{B}$. The equivalent formula in CNF can be obtained by pushing all the negations to the level of variables and distributing all disjunctions over conjunctions.

Given a Boolean formula $f : \mathbb{B}^n \to \mathbb{B}$, we call a *satisfying assignment* of f any tuple of Booleans $(b_1, \dots, b_n) \in \mathbb{B}^n$ such that $f(b_1, \dots, b_n) = 1$. We say that a formula is *satisfiable* if it has at least one satisfying assignment. The satisfying assignments of a given formula f with n variables carve out a subset $[f = 1] := \{(b_1, \dots, b_n) : f(b_1, \dots, b_n) = 1\}$ of $\mathbb{B}^n$.

As mentioned in the introduction, satisfiability – the existence of a satisfying assignment for a given Boolean formula – is a relational constraint. Furthermore, relations tend to admit well-behaved diagrammatic calculi. For these two reasons, we turn to relations as our semantics. The class of relations to consider is naturally suggested to us once we examine more closely the satisfying assignments of clauses, the fundamental building block of CNF formulas. Let $c_{m,n} = \neg x_1 \vee \dots \vee \neg x_m \vee y_1 \vee \dots \vee y_n$ be a clause, with m negative literals and n positive ones. Note the satisfying assignments of $c_{m,n}$ form a subset $[c_{m,n} = 1]$ of $\mathbb{B}^m \times \mathbb{B}^n$ that is compatible with the order over $\mathbb{B}$: if $(a_1, \dots, a_m, b_1, \dots, b_n)$ is a satisfying assignment of $c_{m,n}$, any other assignment $(a'_1, \dots, a'_m, b'_1, \dots, b'_n)$ such that $a'_i \leq a_i$ and $b_j \leq b'_j$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$, is also satisfying. In other words, $[c_{m,n} = 1]$ is an *upward-closed* subset of $(\mathbb{B}^{op})^m \times \mathbb{B}^n$, where the order on the product of posets $X \times Y$ is given by $(x, y) \leq_{X \times Y} (x', y')$ iff $x \leq_X x'$ and $y \leq_Y y'$. For example, the clause $\neg x \vee y_1 \vee y_2$ has as satisfying assignments $(0, (0, 0)), (1, (1, 0)), (1, (0, 1))$ and all tuples that are greater than these in $\mathbb{B}^{op} \times \mathbb{B}^2$. Another example: the set of satisfying assignments of $\neg x \vee y$ is $\{(0, 0), (0, 1), (1, 1)\}$ defines an upward-closed subset of $\mathbb{B}^{op} \times \mathbb{B}$ which we have already encountered as the order on $\mathbb{B}$ itself, *i.e.*, $\{(x, y) \mid x \leq y\}$.

This monotonicity property of clauses and their sets of satisfying assignments suggests the following definition for the kind of relations that we wish to study.

Definition 2.1 A *monotone relation* $R : X \to Y$ between two posets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$ is a relation $R \subseteq X \times Y$ satisfying the following monotonicity condition: if $(x, y) \in R$, then for arbitrary $x' \in X$ and $y' \in Y$, $x' \leq x$ and $y \leq y'$ imply $(x', y') \in R$.

Monotone relations compose as expected: $R \mathbin{;} S := \{(x, z) \mid \exists y \text{ such that } (x, y) \in R, (y, z) \in S\}$. Also, the composition of monotone relations is monotone, so we can organise them into a category.

Proposition 2.2 (see e.g. [16]) *Posets together with monotone relations between them form a category, denoted as* MonRel. *Moreover,* MonRel *equipped with the product of posets and the one-element poset* $\langle \{*\}, \{(*, *)\} \rangle$ *forms a symmetric monoidal category*

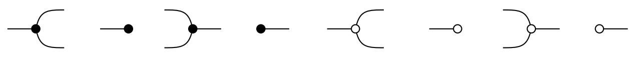
Note that we can turn a monotone function f into the monotone relation $\{(x, y) \mid f(x) \leq y\} \subseteq X \times Y$, so that MonRel contains the category of monotone functions as a subcategory.

Having established their connection to satisfiability, we focus on monotone relations between tuples of Booleans, *i.e.* between products of the two-element Boolean algebra $\mathbb{B}$ (and its dual $\mathbb{B}^{op}$), as our primary semantic objects of interest. In the next two sections, we will develop a diagrammatic calculus axiomatising the full symmetric monoidal subcategory of MonRel spanned by finite Boolean algebras. Moreover, the calculus will shed further light on the relationship between monotonicity and satisfiability. At first glance, from the point of view of satisfiability, it looks like something is lost by restricting our semantics to monotone relations. However, the results of Sections 4 and 5 demonstrate that, perhaps surprisingly, this is not the case. We will come back to this specific point in Remark 4.6.

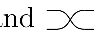
3 The Diagrammatic Calculus

In this section we introduce SATA (SATisfiability Algebra), a diagrammatic calculus for monotone relations. It will consist of a syntax of string diagrams (Definition 3.1) and an inequational theory (Figure 1). The semantic interpretation will be given by a symmetric monoidal functor from the syntax category to the category of monotone relations (Definition 3.2). We will later prove that the inequational theory is *sound and complete* for this interpretation.

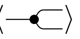
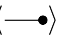
Definition 3.1 We call Syn the strict symmetric monoidal category freely obtained from a single generating object 1 and the following generating morphisms, depicted as string diagrams [28]:

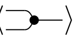
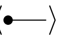
 (1)

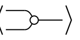
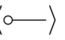
We write SATA for the category Syn quotiented by the axioms of Figure 1. For a review of how to form this quotient and, more broadly, of the general methodology of presenting symmetric monoidal categories via (in)equational theories, we refer the reader to [4], in particular Section 3 of that work.

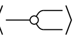
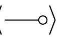
We shall identify the monoidal product $1^{\oplus n}$ of n copies of 1 with the natural number n, and write —— for the identity morphism on 1 and  for the symmetry morphism on 1. We depict the monoidal product $c \oplus d$ of two diagrams as their vertical juxtaposition, and the composition $c\,;d$ as the horizontal concatenation of c and d. We now describe the interpretation of string diagrams as monotone relations.

Definition 3.2 Let $\langle\cdot\rangle$: Syn $\to$ MonRel to be the (lax) symmetric monoidal functor freely obtained by mapping the generating object 1 to $\mathbb{B}$ and the generating morphisms of Syn as follows:

$\langle$$\rangle = \{(x,(y_1,y_2)) \mid x \le y_i,\ \text{and } x,y_1,y_2 \in \{0,1\}\}$ $\langle$$\rangle = \{(x,\bullet) \mid x \in \{0,1\}\}$

$\langle$$\rangle = \{((x_1,x_2),y) \mid x_i \le y,\ \text{and } x_1,x_2,y \in \{0,1\}\}$ $\langle$$\rangle = \{(\bullet,x) \mid x \in \{0,1\}\}$

$\langle$$\rangle = \{((x_1,x_2),y) \mid x_1 \wedge x_2 \le y,\ \text{and } x_1,x_2,y \in \{0,1\}\}$ $\langle$$\rangle = \{(\bullet,1)\}$

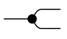
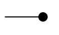
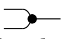
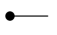
$\langle$$\rangle = \{(x,(y_1,y_2)) \mid x \le y_1 \vee y_2,\ \text{and } x,y_1,y_2 \in \{0,1\}\}$ $\langle$$\rangle = \{(0,\bullet)\}$

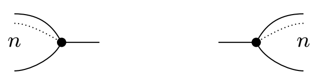
Note that, as $\langle\cdot\rangle$ is freely generated, $\langle$——$\rangle \coloneqq \{(x,y) \mid x \le y \text{ and } x,y \in \{0,1\}\}$, $\langle c\,;d\rangle \coloneqq \langle c\rangle \fatsemi \langle d\rangle$, and $\langle c \oplus d\rangle \coloneqq \langle c\rangle \times \langle d\rangle$.

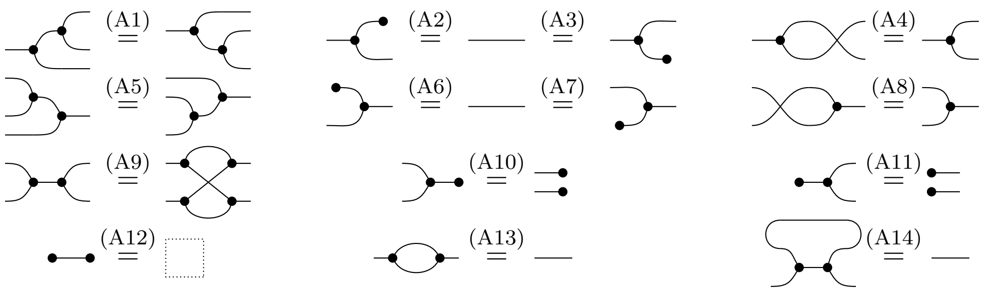
In general, the interpretation provided by $\langle\cdot\rangle$ is not faithful: distinct morphism of Syn may be mapped to the same monotone relation. However, when we quotient the syntax by the inequational theory *SATA* (Figure 1) the mapping SATA $\to$ MonRel we obtain from the quotiented syntax to the semantics is still a symmetric monoidal functor and is now faithful – this is what we mean by soundness (functoriality) and completeness (faithfulness) of *SATA*.

Theorem 3.3 (Soundness and Completeness) *For any $c, d \in$ Syn, $c \le_{\mathsf{SATA}} d$ if and only if $\langle c\rangle \subseteq \langle d\rangle$.*

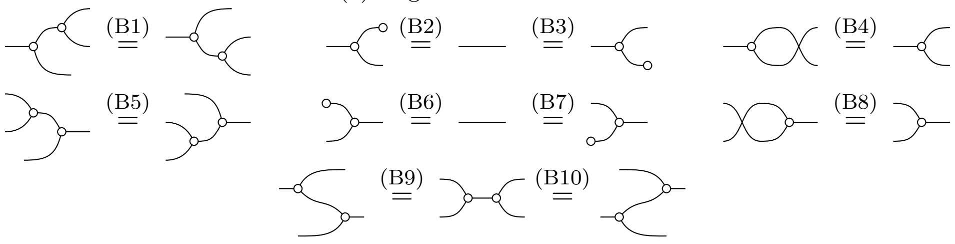
The proof will be given in Section 6. We now highlight some features of *SATA*. They will come handy when using it to study Boolean satisfiability and logic programming, in the next two sections.

- (A1)-(A4) state that  and  form a commutative comonoid, while (A5)-(A8) states that  and  form a commutative monoid. In standard algebraic syntax, we can implicitly associate multiple applications of a monoid operation say, to the left, in order to avoid a flurry of parentheses, *e.g.* $a \vee b \vee c = (a \vee b) \vee c$. Thanks to laws (A1)-(A4), the same principle applies to an associative $2 \to 1$ diagram of *SATA*. This justifies introducing generalised n-ary operations as syntactic sugar, as on the left below. The same works for comonoids, by simply flipping the definitions, as on the right below.

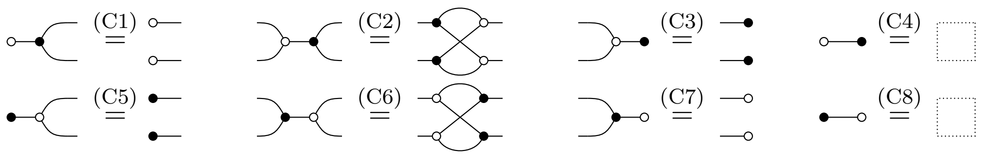




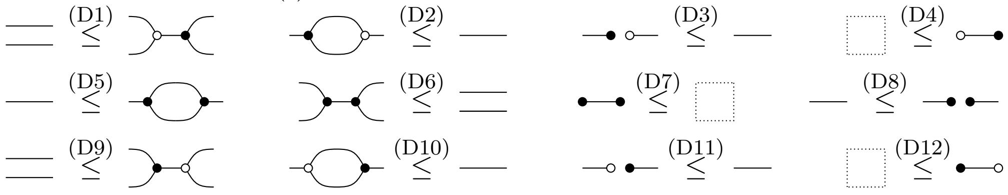
(a) Degenerate bimonoid



(b) Frobenius algebra



(c) White-black and black-white bimonoids



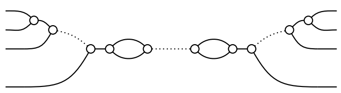
(d) Adjunctions

Fig. 1. The equational theory *SATA*

- (B1)-(B10) state that , , , forms a Frobenius algebra [9]. A first important consequence is that this structure makes SATA a *compact closed category* [21], a category in which one can interpret fixed-points or iteration [18]. Specifically, we have a cup and cap defined as and respectively, satisfying the 'snake' equation = = . Note that axiom (A14) is defined with these as syntactic sugar.

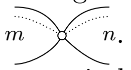
  Moreover, Frobenius algebras greatly reduce the mental load in keeping track of different ways of composing , , , , as they turn out to be equal. This simplification is more formally stated as the *spider theorem* — see e.g. [19, Theorem 5.21].

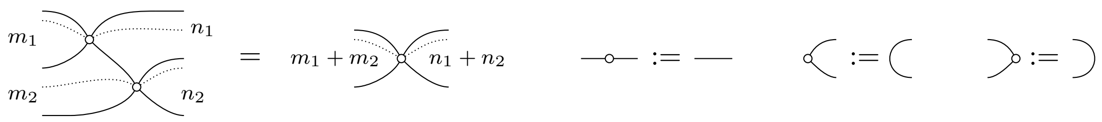
Proposition 3.4 (Spider theorem) *If* , , , *for a commutative Frobenius structure, any* connected *morphism $m \to n$ made out of* , , , , *identities, symmetries, using composition and the monoidal product, is equal to the following normal form*



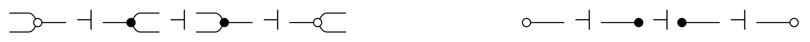
In the statement, the word connected refers to the diagram as a undirected graph: we assume there is at least one path made up of wires, between any two white nodes in the diagram.

In our axiomatic theory, we can simplify the normal form of Theorem 3.4 even further to get rid of loops. Making crucial use of axiom (A14), we can derive =   ($*$) (Corollary B.1 in

Appendix). Note it is unusual for a Frobenius algebra to satisfy $(*)$ as those that have appeared in the literature satisfy a version of $(*)$ with the rhs set to the identity (*e.g.* [29]) or to (*e.g.* [11]). Thanks to the spider theorem and equation $(*)$, we can see any *simply connected* composition of white monoids and comonoids as a single node, whose only relevant structure is its number of dangling wires on the left and on the right. The normal form guarantees that we can unambiguously depict any such morphism $m \to n$ as a *spider* with m wires on the left and n on the right: . With this syntactic sugar, the axioms of the Frobenius structure , , , can be concisely expressed via the following *spider fusion* scheme:



- The C block of axioms state that (, , ,) and (, , ,) form two bimonoids. Semantically, these hold when the underlying poset is a lattice (has binary meets and joins). They also imply that SATA contains as a monoidal subcategory, the category of Boolean matrices with the direct sum as monoidal product (*cf.* Appendix D.2).
- The D axioms present a number of adjunctions in the 2-categorical sense (or rather, Galois connections, because the 2-morphisms are simply inclusions). Two morphisms $f\colon X \to Y$ and $g\colon Y \to X$ are adjoint if $id_X \le f\,;g$ and $g\,;f \le id_Y$. We write this as $f \vdash g$ (f left adjoint to g). The situation can be summarised by the following six adjunctions:



The adjunctions $\dashv$ and $\dashv$ give the defining inequations of cartesian bicategories [9].

To understand where the other adjunctions come from, it is helpful to adopt a semantic perspective. For example, recall that $\langle\;\rangle = \{((x_1,x_2),y) \mid x_1 \wedge x_2 \le y\}$ and $\langle\;\rangle = \{(x,(y_1,y_2)) \mid x \le y_1, x \le y_2\} = \{(x,(y_1,y_2)) \mid x \le y_1 \wedge y_2\}$. Thus, one can see the adjunction $\dashv$ as arising from the duality between the two different ways of turning the monotone function $\wedge : \mathbb{B}^2 \to \mathbb{B}$ into a monotone relation (by placing it on either side of the $\le$ symbol in the corresponding set comprehension).

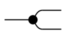
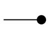
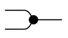
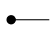
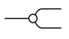
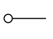
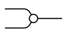
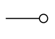
Semantically, they guarantee that the underlying poset $\mathbb{B}$ is a *lattice*, *i.e.*, has binary meets and joins. Together with the Frobenius axioms (B9)-(B10), they imply that $\mathbb{B}$ is a complemented distributive lattice, in other words, a Boolean algebra.

Remark 3.5 Note that we did not aim for a minimal presentation of the theory. There are obvious redundancies: for example, axiom (C4) immediately implies (D4). Similarly, many equations of the B block can be derived from those of the D block and the corresponding properties of the black generators in the A block. We have included these redundancies in order to avoid burdening the reader with too many lemmas deriving equations that we will need often, and to highlight key algebraic structures that occur in related theories (*e.g.* bimonoids). We include some useful derived laws in Appendix B.

4 SATA as the Algebra of Satisfiability

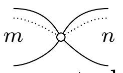
4.1 Picturing SAT

The language we have introduced in the previous section can be seen as a diagrammatic notation to reason about Boolean formulas expressed in conjunctive normal form. As hinted at in the introduction, we can associate to every diagram a CNF formula, with the relational semantics of the diagram given by its set of satisfying assignments. The generators can be interpreted as follows:

 $\mapsto (\neg x \vee y_1) \wedge (\neg x \vee y_2)$	 $\mapsto 1$	 $\mapsto (\neg x_1 \vee y) \wedge (\neg x_2 \vee y)$	 $\mapsto 1$
 $\mapsto \neg x \vee y_1 \vee y_2$	 $\mapsto y$	 $\mapsto \neg x_1 \vee \neg x_2 \vee y$	 $\mapsto \neg x$

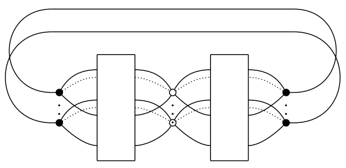
A word of warning: an identity wire does not identify two variables $x = y$; rather, it represents the clause $\neg x \vee y$. The monoidal product acts like conjunction: if d_1 corresponds to some formula f_1 and

d_2 corresponds to some formula f_2, their monoidal product corresponds to $f_1 \wedge f_2$. Composition is more subtle. Given a formula f, let $\exists x f := f[x = 1] \vee f[x = 0]$, where $f[x = b]$ denotes f in which we have replaced all occurrences of x by the Boolean value b. To explain composition, we further assume for simplicity that we compose $c : m \to 1$ and $d : 1 \to n$ along a single wire. Let e and f be their respective associated formulas, and let y be the variable that denotes the single wire in the codomain of c and x the single wire in the domain of d. Then $c\,; d$ corresponds to the formula $\exists x \exists y (e \wedge f) \wedge (x \vee \neg y)$.

One of the key observations is that an arbitrary clause with m negative literals and n positive literals is simply a generalised white node, as introduced above: 

$\mapsto \quad \neg x_1 \vee \cdots \vee \neg x_m \vee y_1 \vee \cdots \vee y_n$.
Note that if we disallow the use of —○⊏, we can only represent clauses with at most one positive literal, better known as *Horn clauses*. We will make extensive use of Horn clauses in the next section, to give a diagrammatic account of propositional logic programs.

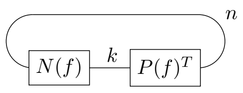
Remark 4.1 We could have given an infinitary presentation of the same symmetric monoidal category with one generator for each clause containing m negative literals and n positive ones (keeping the same black generators). We could give a presentation with a finite number of generators (and inequations), with at most three wires, for the same reason that SAT can be reduced to 3-SAT.



The payoff is that we can use the diagrammatic calculus to encode the satisfiability of Boolean formulas in a very natural way. For a given Boolean formula f in CNF, SAT(f) can be depicted as a closed diagram (one of type $0 \to 0$) of the form on the right, where the white nodes in the middle encode clauses and the black nodes take care of the variable management (copying and deleting) needed to connect variables to the clauses in which they appear, *i.e.* to the corresponding white nodes. Finally, the boxes are just permutations of the wires.

There are - up to equality - only two diagrams of type $0 \to 0$, namely ⬚ (the empty diagram) and ○–○, interpreted as the relations $\{\bullet\}$ and $\varnothing$, respectively. Following the completeness of our equational theory, the soundness of our encoding of SAT implies that a formula f is satisfiable iff the associated diagram is equal to ⬚, as stated in Theorem 4.2 below. Furthermore, using our equational theory, we can derive the (un)satisfiability of a SAT instance purely algebraically, justifying the claim that our diagrammatic calculus captures the algebra of satisfiability.

Let us describe the encoding of SAT(f) in more detail. Given a CNF formula f with n free variables and k clauses (both arbitrarily ordered), let $N(f)$ be the $k \times n$ Boolean matrix where $N(f)_{i,j} = 1$ iff $\neg x_i$ appears in the j-th clauses of f, and $P(f)$ be the $n \times k$ Boolean matrix where $P(f)_{i,j} = 1$ iff x_j appears in the i-th clauses of f. In other words, $N(f)$ encodes negative occurrences of variables, and $P(f)$ their positive occurrences. We use the standard diagrammatic representation of matrices using a commutative bimonoid to define diagrams for $N(f)$ and $P(f)$—a translation which is recalled in Appendix D.2. We use below the same names for the associated diagrams: $N(f)$ denotes the corresponding matrix encoded using —●⊏, —●, ○—, ⊐○—, and $P(f)^T$ the transpose of $P(f)$ encoded using —○⊏, —○, ●—, ⊐●—.

Theorem 4.2 *A CNF formula f is satisfiable if and only if* 

$=$ ⬚.

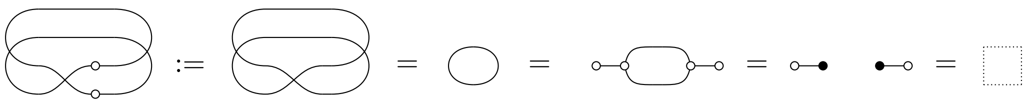
Proof. Call $d(f)$ the leftmost diagram above. Computing its semantics, we get that

$$\langle d(f) \rangle = \{\bullet\} \quad \Leftrightarrow \quad \exists \mathbf{x} \exists \mathbf{y} \in \mathbb{B}^n \left(\mathbf{y} \leq \mathbf{x} \text{ and } N(f)\mathbf{x} \leq P(f)^T \mathbf{y} \right) \quad \Leftrightarrow \quad \exists \mathbf{x} \in \mathbb{B}^n \left(N(f)\mathbf{x} \leq P(f)^T \mathbf{x} \right)$$

If we use negation and disjunction for the Boolean algebra $\mathbb{B}^k$, the last condition can be written as $\exists \mathbf{x} \in \mathbb{B}^n \left(\neg N(f)\mathbf{x} \vee P(f)^T \mathbf{x} \right)$. This is a k-tuple of disjunctions that encode the same clauses as f by construction. Therefore $\neg N(f)\mathbf{x} \vee P(f)^T \mathbf{x}$ is satisfiable iff f is and thus $\langle d(f) \rangle = \{\bullet\}$ iff f is satisfiable. By completeness (Theorem 3.3), we conclude that f is satisfiable iff $d(f) =$ ⬚. □

Example 4.3 $(\neg x \vee y) \wedge (x \vee \neg y)$. Here, we have two variables that appear as negative and positive literals. So we will need two loop wires connecting two bottom wires for $\neg x \vee y$ to two top wires for x, y. Each of these wires correspond to a literal that appears precisely once in one of the two clauses $(\neg x \vee y)$

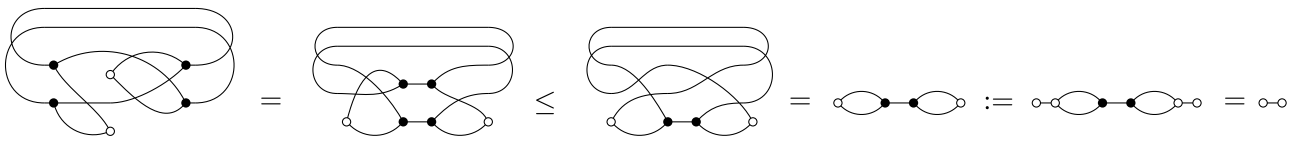
and $(x \vee \neg y)$, so we can just connect them directly - we don't need any white nodes in this case because we just have to connect two wires together. This gives:



which we prove is satisfiable using only the axioms of the equational theory.

Remark 4.4 Inequalities can be used to prove that a formula is satisfiable or unsatisfiable more efficiently. As we have explained above, there are - up to equality - only two diagrams of type $0 \to 0$, namely ⬚ $\leq$ ∘–∘. Thus, if we can show that $d(f) \leq$ ∘–∘ for some formula f, we can strengthen this inequality to an equality and we have shown that f is unsatisfiable. Similarly, if we can show that ⬚ $\leq d(f)$ then $d(f) =$ ⬚ so f is satisfiable.

Example 4.5 $(\neg x \vee y) \wedge (x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$. Using the same principles as above, we get:



As we will explain below, in 4.2, the central inequality can be seen as an instance of *resolution*.

Remark 4.6 [Monotonicity and satisfiability.] It is helpful at this point to pause the development in order to examine the relationship between monotonicity and satisfiability. As we have already pointed out, at first glance, it appears that restricting ourselves to monotone relations is insufficient to study satisfiability. Indeed, monotone formulas (*i.e.* those which do not use negation) are always satisfiable. Correspondingly, at the level of the diagrammatic syntax, we do not have access to a primitive negation operation, which non-monotonic. As a result, we can never explicitly enforce that a variable is the negation of another. So how can we encode SAT without negation? The key idea is that we do have a restricted form of negation, which corresponds to changing the direction of wires, using the cups and caps.

Even though the diagrammatic translation is straightforward, there first appears to be a mismatch between a given CNF formula and the semantics of the diagram representing a SAT instance. A variable appearing as a positive and negative literal in a given formula f, is depicted as two wires – one for its positive occurrences and one for its negative occurrences. When we wish to existentially quantify over a variable of a SAT instance, we simply trace out the corresponding two wires, connecting them. In the relational semantics, the resulting loop joining the two wires represents the additional clause $\neg y \vee x$, where y always appear positively in other clauses, and x always appears negatively. This leaves the possibility of setting both variables to 0, which satisfies $\neg y \vee x$ but does not match the intuition that these two variables should behave like negated version of each other. However, the assignment $y = 0, x = 0$ does not affect the satisfiability of the overall diagram representing the corresponding SAT instance [2]. Indeed, an issue can only arise if the assignment $y = 0, x = 0$ is required to make the diagram satisfiable. But this can never be the case because y always appears positively in any clause other than $\neg y \vee x$. This implies that the satisfiability of the clauses in which y appears will only depend on the other variables and therefore leave the overall satisfiability of the diagram unaffected.

4.2 SAT-solving, diagrammatically?

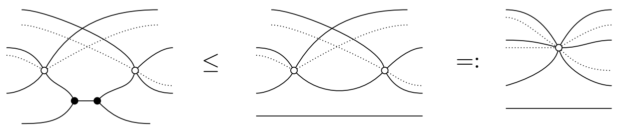
Now that we know how to represent SAT instances in our diagrammatic language, it is useful to give meaning to the axioms in the light of this interpretation. First, the (co)associativity, (co)unitality and (co)commutativity of the various (co)monoids are equivalent to the associativity, unitality and commutativity of $\wedge$ and $\vee$ in propositional logic. The loop removal axiom from Fig. 1a admits a natural interpretation in terms of satisfiability: if $\exists x (C \vee x \vee \neg x)$ for some clause C, we can always satisfy $x \vee \neg x$ and therefore

[2] In light of Theorem 4.2, we say that a closed diagram d is satisfiable when $\langle d \rangle = \{\bullet\}$.

we can remove constraint imposed by the whole clause $C \vee x \vee \neg x$, regardless of the assignment of values to the other variables appearing in C. This axiom allows us to remove this kind of redundant constraints.

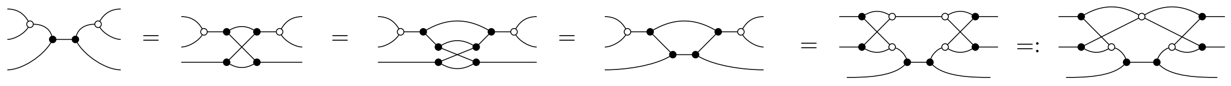
That the white nodes form a Frobenius algebra is more interesting: it stems from the fact that $\exists x.(C \vee x) \wedge (D \vee \neg x)$ allows us to deduce the clause $C \vee D$, and is thus the diagrammatic translation of the *resolution* rule [27] in classical propositional logic!

Resolution has come to occupy a central role in SAT-solving as modern SAT-solvers can be thought of as resolution proof systems. It is well-known that solvers based on a variant of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [15,14] implement a form of *tree-like* resolution, while those that are augmented with clause learning (*e.g.* CDCL) correspond to more general forms of resolution proofs, that allow for the reuse of learned clauses [26]. Tree-like resolution proofs can be easily interpreted diagrammatically. Recall from our interpretation of SAT instances that clauses are simply white nodes whose left (*resp.* right) wires encode negative (*resp.* positive) literals. Thus, if we have two clauses $C \vee x$ and $\neg x \vee D$, the corresponding diagram will contain a subdiagram equal to the leftmost one in (2) below. The black nodes in the subdiagram represent the fact that x might also appear (negatively or positively) in other clauses. From this, resolution allows us to deduce $C \vee D$; tree-like resolution proofs further impose that, if we apply a deduction step, we do not use the original clauses $C \vee x$ and $\neg x \vee D$ again. This has a simple diagrammatic counterpart—we can apply axiom (D6) to connect the two white nodes corresponding to the two clauses in premise of a resolution step directly, effectively replacing them with a single node that represents the learned clause:


(2)

The remaining wire for other uses of x is unaffected. We have already used the same idea in Example 4.5 above, in order to resolve the subdiagrams corresponding to the clauses $\neg x \vee y$ and $x \vee y$.

General resolution – where we are allowed to keep the two clauses in the premise of the rule – also has a diagrammatic counterpart. In fact, it is implemented as a sequence of equalities (instead of the inequality we have used above to interpret tree-like resolution), reflecting the fact that the process does not forget any information. The idea is to first copy the two clauses $C \vee x$ and $\neg x \vee D$ to leave them available for any further resolution with other clauses that use the variable x. We illustrate this process with two clauses (containing only three literals for simplicity this time) below:



In the resulting diagram, the quaternary white node represents the clause $C \vee D$, while the other two represent the original clauses C and D.

5 SATA as a proof system for logic programming

In this section we turn to logic programming. SATA allows us to compute the usual semantics of propositional definite logic programs via diagrammatic equational reasoning (Theorem 5.6). More importantly, we obtain a decidable, purely equational procedure to decide the equivalence of program, by proving the equivalence of their representing diagrams in SATA (Theorem 5.7).

First we briefly recall the basics of definite logic programming, and refer the readers to [23] for details. Logic programming is a programming paradigm which reduces computation to proof search in formal logic. It is widely used in knowledge representation [1], database theory [10], constraint solving [20]. We focus on the simplest yet prototype class of logic programs, called *propositional definite logic programs*, formally defined as follows. We fix some finite set of atoms $At = \{a_1, \dots, a_k\}$. A *Horn clause* ψ is a formula of the form $b_1, \dots, b_k \to a$, where $a, b_1, \dots, b_k \in At$ and $b_1 \dots, b_k$ are distinct. Here the atom a and the set $\{b_1, \dots, b_k\}$ are called the *head* ($\mathsf{head}(\psi)$) and the *body* ($\mathsf{body}(\psi)$) of the clause, respectively. If a clause has an empty body, then it is also called a *fact*. A *propositional definite logic program* (or simply *logic program*) is a finite set of Horn clauses based on At. Intuitively, a clause $b_1, \dots, b_k \to a$ reads 'if $b_1, \dots, b_k$

hold, then a also holds', which gives the intuitive meaning of a program $\mathbb{L}$ to be the set of all atoms that are deducible from $\mathbb{L}$ in finitely many steps. This is formally defined as the *least Herbrand model semantics*. An *interpretation* $\mathcal{I}$ over At is an assignment $\mathcal{I}\colon At \to \mathbb{B}$. For simplicity, we will also write an interpretation $\mathcal{I}$ over At as a subset $\{a \in At \mid \mathcal{I}(a) = 1\}$.

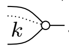
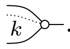
A logic program $\mathbb{L}$ defines an *immediate consequence operator* $\mathbf{T}_{\mathbb{L}}\colon \mathbb{B}^{At} \to \mathbb{B}^{At}$ such that, given an interpretation $\mathcal{I}$,

$$\mathbf{T}_{\mathbb{L}}(\mathcal{I})(a) = \begin{cases} 1 & \text{there exists } \mathbb{L}\text{-clause } b_1, \ldots, b_k \to a \text{ such that } \mathcal{I}(b_1) = \cdots = \mathcal{I}(b_k) = 1 \\ 0 & \text{otherwise} \end{cases}$$

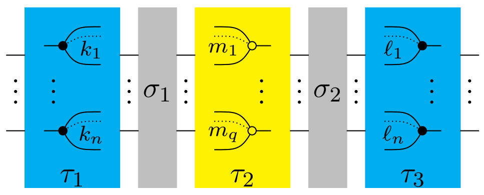
The *Herbrand model* (denoted as $\mathbf{M}_{\mathbb{L}}$) is then the least fixed point of $\mathbf{T}_{\mathbb{L}}$. Such least fixed point always exists because $\mathbb{B}^{At}$ carries a complete lattice structure, and $\mathbf{T}_{\mathbb{L}}$ is monotone on it. We use the following notation: $\wedge$ and $\vee$ are defined pointwise on At, $\mathbf{1}$ and $\mathbf{0}$ are the constant functions mapping arbitrary $a \in At$ to 1 and 0, respectively.

From a deductive perspective, a logic program $\mathbb{L}$ can be viewed as a black box: it takes a set of atoms A as input, and outputs the set of atoms that are deducible from A using $\mathbb{L}$. This is formalised as the *consequence operator* $\mathbf{C}_{\mathbb{L}}\colon \mathbb{B}^{At} \to \mathbb{B}^{At}$ of program $\mathbb{L}$, such that $\mathbf{C}_{\mathbb{L}}(A)$ is the least Herbrand model of the program $\mathbb{L} \cup \{\to a \mid a \in A\}$. In particular, $\mathbf{C}_{\mathbb{L}}(\varnothing)$ is exactly $\mathbf{M}_{\mathbb{L}}$.

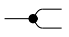
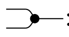
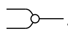
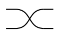
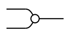
Example 5.1 Consider the program $\mathbb{P} = \{\to a; b \to d; c \to d; c, d \to b\}$ over $At = \{a, b, c, d\}$. Its least Herbrand model $\mathbf{M}_{\mathbb{P}}$ is $\{a\}$. The immediate consequence operator $\mathbf{T}_{\mathbb{P}}$ maps, for instance, $\{c\}$ to $\{a, d\}$. The consequence operator $\mathbf{C}_{\mathbb{P}}$ maps, for instance, $\{c\}$ to $\{a, b, c, d\}$.

We will represent (immediate) consequence operators of logic programs via our diagrammatic language. Immediate consequence operators have been studied from a string diagram perspective in [17]. Compared with (3) below, there each clause is represented by a box, while our diagrammatic language enables us to 'open the boxes' by representing a definite clause as . The key observation is the intuitive reading of clause $b_1, \ldots, b_k \to a$ as $b_1 \wedge \cdots \wedge b_k \leq a$, thus representable as the Syn-morphism .

Definition 5.2 The *immediate consequence diagram* $\mathsf{T}_{\mathbb{L}}$ for program $\mathbb{L}$ is an $n \to n$ morphism defined as follows, where the i-th input and output wires stands for atom a_i, for $i = 1, \ldots, n$.

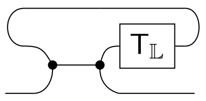


(3)

The two blue blocks τ_1 and τ_3 consist respectively of copiers  and cocopiers : k_i and ℓ_i are the numbers of appearance of atom a_i in all the *bodies* and heads of $\mathbb{L}$-clauses, respectively. The yellow block τ_2 is the parallel composition of , where q is the number of clauses in $\mathbb{L}$, and for each clause ψ_j, m_j is the size of $\mathsf{body}(\psi_j)$. The grey blocks σ_1 and σ_2 consist of appropriately many  that change the wire orders to match the domains and codomains of the blue and yellow blocks. In particular, the i-th input (resp. output) wire is connected to the j-th  if $a_i \in \mathsf{body}(\psi_j)$ (resp. $a_i = \mathsf{head}(\psi)$).

Based on this, we can compositionally represent the consequence operator $\mathbf{T}_{\mathbb{L}}$.

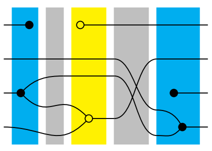
Definition 5.3 The *consequence diagram* for program $\mathbb{L}$ is a Syn-morphism $\mathsf{C}_{\mathbb{L}}\colon n \to n$ defined on the right, where $\mathsf{T}_{\mathbb{L}}$ is the immediate consequence diagram from Definition 5.2.



Note that the interpretation $\langle \mathsf{T}_{\mathbb{L}} \rangle$ is not exactly the operator $\mathbf{T}_{\mathbb{L}}$: the former is a monotone relation while the latter is a monotone function. Nevertheless we say a monotone relation $R \subseteq X \times Y$ *represents* a monotone function $f\colon X \to Y$ if for arbitrary $x \in X$ and $y \in Y$, $(x, y) \in R$ if and only if $f(x) \leq y$. Crucially, if monotone relations R and S represent monotone functions f and g respectively, then $f = g$ if and only if $R = S$.

Proposition 5.4 *The monotone relation* $\langle \mathsf{T}_{\mathbb{L}} \rangle$ *(resp.* $\langle \mathsf{C}_{\mathbb{L}} \rangle$*) represents the operator* $\mathbf{T}_{\mathbb{L}}$ *(resp.* $\mathbf{C}_{\mathbb{L}}$*).*

Example 5.5 Recall $\mathbb{P}$ from Example 5.1. $\mathsf{T}_{\mathbb{P}}$ is given below left, coloured as in Definition 5.2.



Now we are ready to present the two main results for logic programs. First, *SATA* provides a diagrammatic calculus to compute $\mathbf{C}_{\mathbb{L}}$, in particular to calculate the least Herbrand model as $\mathbf{C}_{\varnothing}$. One can represent an interpretation $\mathcal{I} \in \{0,1\}^{At}$ as a Syn-morphism $d^{\mathcal{I}} := d_1^{\mathcal{I}} \oplus \cdots \oplus d_n^{\mathcal{I}}$ of type $0 \to n$, where each $d_i^{\mathcal{I}} =$ •— if $\mathcal{I}(i) = 0$, and $d_i^{\mathcal{I}} =$ ∘— if $\mathcal{I}(i) = 1$.

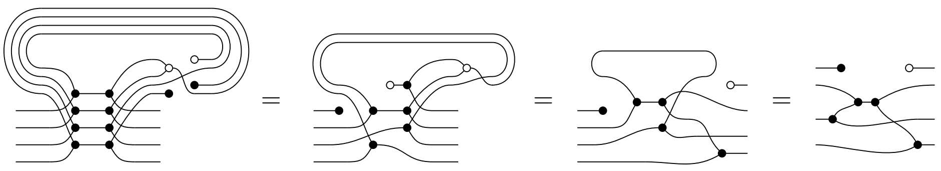
Theorem 5.6 $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mathcal{J}$ *if and only if* $d^{\mathcal{I}} \,;\mathsf{C}_{\mathbb{L}} = d^{\mathcal{J}}$*.*

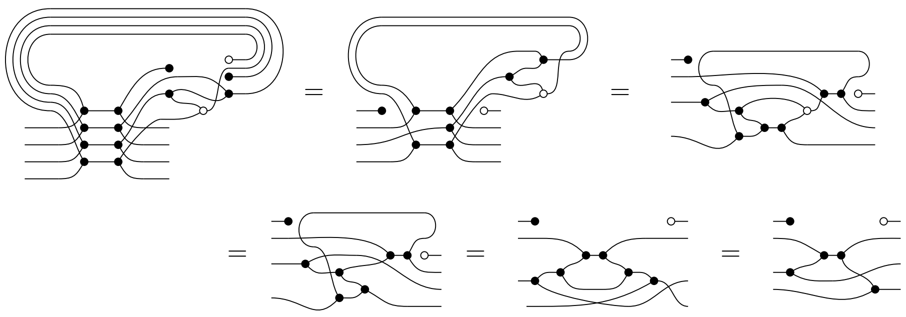
Second, we can turn the problem of the equivalence of two logic programs into proving whether their consequence operator diagrams are equivalent. By Proposition 5.4 and Theorem 3.3, we have:

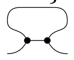
Theorem 5.7 *Given two logic programs* $\mathbb{L}_1$ *and* $\mathbb{L}_2$ *on At,* $\mathbf{C}_{\mathbb{L}_1} = \mathbf{C}_{\mathbb{L}_2}$ *if and only if* $\mathsf{C}_{\mathbb{L}_1} = \mathsf{C}_{\mathbb{L}_2}$*.*

It is worth observing that, as we will see in Section 6, the completeness proof is essentially an algorithm that rewrites an arbitrary Syn-morphism into a normal form diagram (Definition 6.12). This means that the procedure described in Theorem 5.7 to establish equivalence of logic programs in terms of consequence operators is in fact decidable.

Example 5.8 Recall $\mathbb{P}$ from Example 5.1, and let $\mathbb{Q} = \{\to a; a, b \to d; c \to b.\}$. We claim that $\mathbb{P}$ and $\mathbb{Q}$ are equivalent with respect to consequence operator. One can show this by proving the equivalence of their consequence diagrams, namely $\mathsf{C}_{\mathbb{P}} = \mathsf{C}_{\mathbb{Q}}$:

$\mathsf{C}_{\mathbb{Q}} =$ 

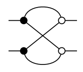
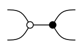
$\mathsf{C}_{\mathbb{P}} =$ 

Remark 5.9 It is helpful to point out that if we interpret Syn in the category $\mathbf{Rel}$ of relations (via a functor $[-] \colon \mathsf{Syn} \to \mathbf{Rel}$), then $\mathsf{C}_{\mathbb{L}}$ — an intuitive candidate to represent consequence operators — is not interpreted as $\{(\mathcal{I}, \mathcal{J}) \mid \mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mathcal{J}\}$. In short, for the semantics to be correct, one has to define [—•⊂] $= \{(x, (x, x)) \mid x \in \mathbb{B}\}$ and [⊃•—] $= \{((x_1, x_2), y) \mid x_1, x_2, y \in \mathbb{B}, x_1 \vee x_2 = y\}$. But then the program $\mathbb{L} = \{a \to a\}$ has $\mathsf{C}_{\mathbb{L}} =$ , and $[\mathsf{C}_{\mathbb{L}}] = \{(x, y) \mid x, y \in \mathbb{B}, x \leq y\} \neq \{(x, x) \mid x \in \mathbb{B}\} = \mathbf{C}_{\mathbb{L}}$.

6 Soundness and Completeness

This section is devoted to the proof of Theorem 3.3, i.e. that the equational theory *SATA* is a complete axiomatisation of monotone relations over (finite) Boolean algebras. Omitted proofs of intermediate results can be found in Appendix D.

The 'only if' direction (soundness) of Theorem 3.3 is straightforward: we just need to show that each axiom is a valid semantic (in)equality. We take (C2) as an example, but will omit the verification of the other axioms:

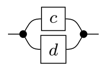
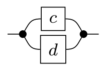
$$\left\langle$$  $$\right\rangle = \{((x_1,x_2),(y_1,y_2)) \in \mathbb{B}^4 \mid \exists z_1,z_2,z_3,z_4 \in \mathbb{B} : x_1 \leq z_1 \wedge z_2, x_2 \leq z_3 \wedge z_4, z_1 \wedge z_3 \leq y_1, z_2 \wedge z_4 \leq y_2\}$$
$$= \{((x_1,x_2),(y_1,y_2)) \in \mathbb{B}^4 \mid \exists z_3,z_4 \in \mathbb{B} : x_2 \leq z_3 \wedge z_4, x_1 \wedge z_3 \leq y_1, x_1 \wedge z_4 \leq y_2\}$$
$$= \{((x_1,x_2),(y_1,y_2)) \in \mathbb{B}^4 \mid \exists z_3,z_4 \in \mathbb{B} : x_1 \wedge x_2 \leq y_1, x_1 \wedge x_2 \leq y_2\}$$
$$= \{((x_1,x_2),(y_1,y_2)) \in \mathbb{B}^4 \mid x_1 \wedge x_2 \leq y_1 \wedge y_2\} = \langle$$  $$\rangle$$

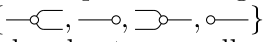
We now focus on the 'if' direction (completeness). Before the technical developments, we provide a roadmap of the proof. We first show that we can make two simplifying assumptions: (1) the completeness statement about inequalities/inclusions can be reduced to one that involves only equalities (Lemma 6.1); (2) We simplify the problem further by showing that we can restrict the proof of completeness to diagrams $m \to 0$ without loss of generality (Lemma 6.2).

To prove completeness for $m \to 0$ diagram, we adopt a normal form argument that is common for the completeness proof of many diagrammatic calculi [24,25]. In a nutshell, a normal form defines a certain syntactic representative for each semantic object in the image of the interpretation, and shows that if two diagrams have the same semantics, then they are both equal in *SATA* to this syntactic representative. The structure of the normal form argument is as follows.

- We give a procedure to rewrite any diagram into one in *pre-normal form*, using equations of *SATA* (Lemma 6.3). Intuitively, pre-normal form diagrams represent sets of clauses of the form $\neg x_1 \vee \cdots \vee \neg x_n$.
- Each downward closed subset is the set of satisfying assignment for a minimal set of such clauses (Lemma 6.9). The diagrammatic counterpart of this minimal set of clauses is our chosen normal form for each equivalence class of diagrams (Definition 6.12). We show that every diagram in pre-normal form is equal to one in normal form (Lemma 6.18).
- Finally, since every semantic object has a canonical diagrammatic representative – a normal form diagram – completeness follows from the fact that two diagrams having the same semantics have the same normal form (Proposition 6.14).

As explained above, completeness for equalities is enough to derive that for inequalities.

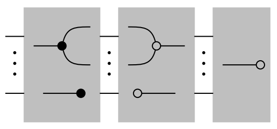
Lemma 6.1 *For diagrams c, d, (i) if*  *$= c$, then $c \leq d$ and (ii) if $\langle c\rangle \subseteq \langle d\rangle$ then* $\left\langle$  $\right\rangle = \langle c\rangle$.

In addition, we can restrict the completeness proof to diagrams with codomain 0. This is the consequence of axioms (B9)-(B10) which endow $\{$  $\}$ with the structure of a Frobenius algebra and SATA with the structure of a compact closed category, allowing us to move wires from the left to the right and vice-versa.

Lemma 6.2 *For arbitrary natural numbers m, n,* $\mathsf{Syn}[m,n] \simeq \mathsf{Syn}[m+n,0]$.

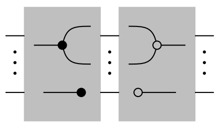
6.1 Pre-normal form

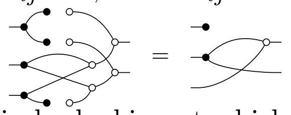
In this subsection, we show that every diagram is equivalent to one of *pre-normal form*, that is, which factorises as on the right. We then explore their close connection with certain Boolean clauses, a key tool in completeness proof. Let us explain how to interpret the factorisation: each grey block represents a diagram which is formed only using the specified subset of generators (using both composition and monoidal product), possibly including some permutations of the wires. The equational theory allows us to rewrite any diagram to pre-normal form.



Lemma 6.3 (Pre-normal form) *Every diagram $m \to 0$ is equal to one of pre-normal form.*

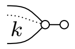
The first two blocks of pre-normal forms deserve special attention. We say that d is a *matrix diagram* if it factorises as on the right. In our semantics, matrix diagrams $m \to n$ denote relations that are the satisfying assignments of sets of



Horn clauses, *i.e.* clauses with at most one positive (unnegated) literal. For each Horn clause, the negated literals correspond to a subset of the left wires, while the only positive literal corresponds to the right wire to which they are connected. Intuitively, matrix diagrams can be thought of simply as matrices with Boolean coefficients: each row encodes the negative literals of a Horn clause through its coefficients. An arbitrary $n \times m$ matrix A can be represented by a matrix diagram with m wires on the left and n wires on the right—the left ports can be interpreted as the columns and the right ports as the rows of A (*cf.* [29, §3.2]). The j-th wire on the left is connected to the i-th wire on the right whenever $A_{ij} = 1$; when $A_{ij} = 0$ they remain disconnected. For example, the matrix $A = \left(\begin{smallmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \end{smallmatrix}\right)$ can be depicted as  . Conversely, given a diagram of this form, we can recover the corresponding matrix by looking at which ports are connected to which. The equational theory *SATA* is complete for the monoidal subcategory generated by —●⊂, —●, ○—, ⊃○—.

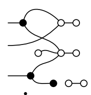
Proposition 6.4 (Matrix completeness) *Let $c, d\colon m \to n$ be two diagrams formed only of the generators —●⊂, —●, ○—, ⊃○—. We have $\langle c \rangle = \langle d \rangle$ iff $c = d$ in SATA.*

We can extract a unique matrix from a pre-normal form diagram, which we will call its *representing matrix*. Moreover, Proposition 6.4 allows us to simplify our reasoning in the proof of completeness: in light of this result, we will assume – wherever it is convenient – that we can identify any two diagrams formed of —●⊂, —●, ○—, ⊃○— that represent the same matrix. This often means that we can be as lax as necessary when identifying diagrams modulo the (co)associativity of —●⊂ or ⊃○— and the (co)unitality of (—●⊂, —●) and (⊃○—, ○—). In particular, given a Boolean matrix A, we can define a unique (up to equality in *SATA*) pre-normal form diagram with representing matrix A.

Pre-normal forms are closely related to certain sets of Boolean clauses which we now introduce. We fix a finite set of variables $Var = \{x_1, \ldots, x_m\}$. By analogy with the corresponding terminology for Boolean formulas (*cf.* Section 2), we call a subset of Var a *¬-clause*. In the rest of this section, a ¬-clause $\{x_{i_1}, \cdots, x_{i_k}\}$ is intended to represent the diagram  whose semantics is given by $\{x_{i_1}, \cdots, x_{i_k} \mid x_{i_1} \wedge \cdots \wedge x_{i_k} \leq 0\}$ which is equal to the set of satisfying assignments of $\neg x_{i_1} \vee \cdots \vee \neg x_{i_k}$ [3]. For conciseness, we sometimes write an assignment $\mathbf{s}\colon Var \to \mathbb{B}$ as an element of $\mathbb{B}^m$, whose i-th element is $\mathbf{s}(i+1)$, for $i = 0, \ldots, m-1$. Given a ¬-clause $\varphi = \{x_{i_1}, \cdots, x_{i_k}\}$, we say that an assignment $(b_1, \ldots, b_m)$ satisfies φ if $\neg b_{i_1} \vee \cdots \vee \neg b_{i_k} = 1$ (equivalently, if $b_{i_1} \wedge \cdots \wedge b_{i_k} = 0$). We say that it satisfies a set of ¬-clause if it satisfies all ¬-clause in the set simultaneously.

From pre-normal form to ¬-clauses Pre-normal form diagrams matter because one can derive a set of ¬-clauses from them, which is closely related with their semantics (Proposition 6.14). Given some pre-normal form diagram $d\colon m \to 0$, we define $\mathsf{clause}(d)$ to be the set of ¬-clauses over m variables as follows. Each row r of the representing matrix A of d generates a ¬-clause φ such that $x_i \in \varphi$ if and only if $r[i] = 1$, and $\mathsf{clause}(d)$ is the set of all ¬-clauses generated by the rows of A. The diagram d and the set of ¬-clauses $\mathsf{clause}(d)$ are semantically equivalent in the following sense.

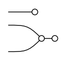
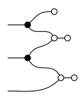
Proposition 6.5 *Let $d\colon m \to 0$ be a pre-normal form diagram. Then $\langle d \rangle$ is exactly the set of all satisfying assignments of* $\mathsf{clause}(d)$*, the le set of ¬-clauses generated by d.*

Example 6.6 The diagram $d =$  , is in pre-normal form and $\mathsf{clause}(d) = \{\{x_1, x_2\}, \{x_1, x_3\}, \{\}\}$. $\langle d \rangle = \varnothing$, the set of all satisfying assignments of $\mathsf{clause}(d)$.

From ¬-clauses to pre-normal form Conversely, given a set of ¬-clauses Φ over Var, we can define a diagram $\mathsf{diag}(\Phi)\colon m \to 0$ in pre-normal form. Assume that $|\Phi| = n$, and we list the ¬-clauses in Φ using lexicographical order as $\varphi_1, \ldots, \varphi_n$ (for instance, $\{x_1, x_3\}$ appears before $\{x_2\}$). Then $\mathsf{diag}(\Phi)$ is defined as the pre-normal form diagram with representing $n \times m$ Boolean matrix A given by $A_{ij} = 1$ precisely if $x_j \in \varphi_i$. Again, Φ and $\mathsf{diag}(\Phi)$ are semantically equivalent in the following sense.

[3] Strictly speaking, because all literals are negative, we should write $\{\neg x_{i_1}, \cdots, \neg x_{i_k}\}$ for the corresponding clause. However, to save the reader a flurry of negations, we will simply write it as the set $\{x_{i_1}, \cdots, x_{i_k}\}$.

Lemma 6.7 *For Φ a set of $\neg$-clauses over Var, $\langle \mathsf{diag}(\Phi)\rangle$ is exactly the set of satisfying assignments of Φ.*

Example 6.8 Let $\mathit{Var} = \{x_1, x_2, x_3\}$, and consider the following two sets of $\neg$-clauses: $\Gamma = \{\{x_1\}, \{x_2, x_3\}\}$ and $\Phi = \{\{x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}$ They have the same set of satisfying assignments $A = \{000, 010, 001\}$; their associated pre-normal form diagrams are $\mathsf{diag}(\Gamma) =$ , $\mathsf{diag}(\Phi) =$ , which verify $\langle \mathsf{diag}(\Gamma)\rangle = \langle \mathsf{diag}(\Phi)\rangle = A$. Moreover, Φ contains the 'redundant' $\neg$-clause $\{x_1, x_2\}$ because the clause $\{x_1\}$ already implies that $\{x_1, x_2\}$ is satisfied (if $\neg x_1 = 1$, then $\neg x_1 \vee \neg x_2 = 0$), while Γ contains no such redundancies. Soon we shall see that Γ is the smallest set of $\neg$-clauses whose set of satisfying assignments is A, and $\mathsf{diag}(\Gamma)$ is in normal form.

6.2 *Normal form and completeness.*

Recall that for completeness we want to select one representative diagram – in normal form – for each semantic object. We use sets of $\neg$-clauses as an intermediate step and choose such representative to be the *minimal set of clauses* among those with a given set of satisfying assignments: a set of $\neg$-clauses Φ is *minimal* if dropping any $\neg$-clause from Φ returns a set Φ' whose set of satisfying assignments is a proper superset of those of Φ. Minimal sets of $\neg$-clauses are unique, thus every given set of satisfying assignment for some set of $\neg$-clauses has a least set of $\neg$-clauses.

Lemma 6.9 *If two minimal sets of $\neg$-clauses Σ and Γ have the same satisfying assignments, then $\Sigma = \Gamma$.*

Lemma 6.10 *A subset of $\mathbb{B}^{\mathit{Var}}$ is downward closed if and only if it is the satisfying assignment of some set of $\neg$-clauses over Var.*

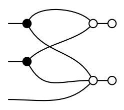
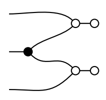
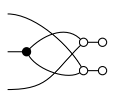
Lemmas 6.9-6.10 imply the existence of a unique minimal set of $\neg$-clauses for each semantic object.

Lemma 6.11 *For every downward closed subset A of $\mathbb{B}^{\mathit{Var}}$ there is a unique minimal set of $\neg$-clauses whose solution set is A.*

The normal form diagrams are defined as the diagrammatic counterpart of minimal sets of $\neg$-clauses.

Definition 6.12 Suppose $d\colon m \to 0$ is a pre-normal form diagram with representing matrix A. We say d is of *normal form* if there are no two distinct rows i, j of A such that $A_{ik} \leq A_{jk}$ for all $k = 1, \ldots, m$.

Note that swapping two rows in the representing matrix A of diagram d does not change $\mathsf{clause}(d)$ (the set of $\neg$-clauses generated by d), so what we really care about is normal form diagrams up to some permutation of the rows of their representing matrices. Let d and e be two normal form diagrams with representing $n \times m$ matrices A and B, respectively. d and e are *equivalent up to commutativity* if A is B with its rows permuted: for all $i \in \{1, \ldots, n\}$, there exists some $j \in \{1, \ldots, n\}$ such that $A_{ik} = B_{jk}$ for all $k = 1, \ldots, m$, and vice versa.

$c_1 =$  $c_2 =$  $c_3 =$ 

Example 6.13 Consider the pre-normal form diagrams above right. c_1 is not of normal form because it represents the matrix $\left(\begin{smallmatrix}1&1&0\\1&1&1\end{smallmatrix}\right)$. c_2 and c_3 are both of normal form, representing matrices $\left(\begin{smallmatrix}1&1&0\\0&1&1\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}0&1&1\\1&1&0\end{smallmatrix}\right)$ respectively, thus equivalent up to commutativity.

Proposition 6.14 *There is a 1-1 correspondence between the following three sets:*

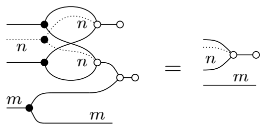
(i) *downward closed subsets of $\mathbb{B}^m$;*

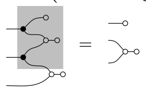
(ii) *minimal sets of $\neg$-clauses over $\mathit{Var} = \{x_1, \ldots, x_m\}$;*

(iii) *normal form diagrams of type $m \to 0$, modulo equivalence up to commutativity.*

Remark 6.15 The proposition above implies a stronger result plain completeness : *SATA* is *fully* complete for the given semantics, as every monotone relation between tuples of Booleans is in the image of the interpretation functor $\langle\cdot\rangle$.

Having established the connection between semantic objects and normal form diagrams, we show that indeed every Syn-morphism is equal to one in normal form. Crucially we need the following lemma, which allows us to remove redundant clauses from a diagram in pre-normal form to obtain one in normal form: if some $\neg$-clause ϕ is strictly larger than another ψ, then one can drop ϕ from Φ and the resulting set of $\neg$-clauses has the same set of satisfying assignments with the original Φ.

Lemma 6.16 *For arbitrary natural numbers m, n,* 

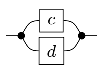
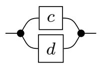
Example 6.17 Recall the two diagrams $\mathsf{diag}(\Gamma)$ and $\mathsf{diag}(\Phi)$ from Example 6.8, and we note that $\mathsf{diag}(\Phi)$ is in normal form while $\mathsf{diag}(\Gamma)$ is not. Yet we can apply Lemma 6.16 (to the grey block) to rewrite $\mathsf{diag}(\Gamma)$ (on the lhs) into the normal form diagram $\mathsf{diag}(\Phi)$ (on the rhs): 

Lemma 6.18 (Normal form) *Every diagram $d \colon m \to 0$ is equal to a diagram in normal form.*

Finally we are ready to prove the completeness part of Theorem 3.3. We start with the following completeness statement (involving only equalities), which follows from Proposition 6.14 and Lemma 6.18.

Lemma 6.19 *For any diagrams $c, d \colon m \to n$, if $\langle c \rangle = \langle d \rangle$, then $c = d$.*

We can then derive completeness from the following line of reasoning.

Proof of Theorem 3.3. We assume that $\langle c \rangle \subseteq \langle d \rangle$. By Lemma 6.1*(ii)*, $\Bigg\langle$  $\Bigg\rangle = \langle c \rangle$. By Lemma 6.19, this implies  $= c$, thus $c \leq d$ according to Lemma 6.1 *(i)*. □

7 Conclusion

An alternative approach to the problem of Boolean satisfiability is to consider the symmetric monoidal category of *all* relations between tuples of Booleans. A close cousin of this category – that of spans between finite sets with cardinality 2^n – was axiomatised in [12]. Adding a single axiom (to enforce transitivity of the order on $\mathbb{B}$) should be sufficient to obtain a complete axiomatisation of the corresponding category of relations. The resulting calculus would certainly be expressive enough to encode SAT instances and derive their (un)satisfiability equationally. However, the equational theory is more complex (reflecting the larger semantic domain), with two distinct Frobenius structures. Moreover, the monotonicity of our semantics, while seemingly limiting at first, allows us to *topologise* certain algebraic aspects of SAT: negation is represented as a change of the direction of wires, an essential feature of our calculus. These choices result in a tighter correspondence with formulas in CNF and a simpler equational theory, with close links to existing SAT-solving algorithms, as we have begun to investigate in Section 4.2. We hope to explore these connections further in future work, as well as the possibility of devising new heuristics for SAT-solving guided by the equational theory.

On the logic programming side, the current paper extends work from [17], in which two of the authors of the current paper propose a functorial semantics for logic programs. Using the completeness result of the paper, we are able to go further to obtain a diagrammatic calculus to compute Herbrand semantics of logic programs, and show two programs are equivalent by proving the equivalence of two diagrams. There are two main directions of further exploration. The first is to extend the current framework to include more logical components, for example to consider logic programming admitting variables and/or negation. The second is to invent a diagrammatic calculus for probabilistic and weighted logic programming.

References

[1] Baral, C. and M. Gelfond, *Logic programming and knowledge representation*, The Journal of Logic Programming **19** (1994), pp. 73–148.

[2] Boisseau, G. and R. Piedeleu, *Graphical piecewise-linear algebra*, in: *International Conference on Foundations of Software Science and Computation Structures*, Springer, Cham, 2022, pp. 101–119.

[3] Bonchi, F., A. Di Giorgio and P. Sobocinski, *Diagrammatic polyhedral algebra*, arXiv preprint arXiv:2105.10946 (2021).

[4] Bonchi, F., D. Pavlovic and P. Sobocinski, *Functorial semantics for relational theories*, arXiv preprint arXiv:1711.08699 (2017).

[5] Bonchi, F., R. Piedeleu, P. Sobociński and F. Zanasi, *Graphical affine algebra*, in: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019, pp. 1–12.

[6] Bonchi, F., J. Seeber and P. Sobociński, *Graphical conjunctive queries*, Computer Science Logic 2018 (2018).

[7] Bonchi, F., P. Sobociński and F. Zanasi, *Interacting hopf algebras*, Journal of Pure and Applied Algebra **221** (2017), pp. 144–184.

[8] Boole, G., "An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities," Dover, 1854.

[9] Carboni, A. and R. F. Walters, *Cartesian bicategories i*, Journal of pure and applied algebra **49** (1987), pp. 11–32.

[10] Ceri, S., G. Gottlob, L. Tanca et al., *What you always wanted to know about datalog(and never dared to ask)*, IEEE transactions on knowledge and data engineering **1** (1989), pp. 146–166.

[11] Coecke, B. and A. Kissinger, *The compositional structure of multipartite quantum entanglement*, in: *International Colloquium on Automata, Languages, and Programming*, Springer, 2010, pp. 297–308.

[12] Comfort, C., *The zx&-calculus: A complete graphical calculus for classical circuits using spiders*, arXiv preprint arXiv:2004.05287 (2020).

[13] Cook, S. A., *The complexity of theorem-proving procedures*, in: *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[14] Davis, M., G. Logemann and D. Loveland, *A machine program for theorem-proving*, Communications of the ACM **5** (1962), pp. 394–397.

[15] Davis, M. and H. Putnam, *A computing procedure for quantification theory*, Journal of the ACM (JACM) **7** (1960), pp. 201–215.

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

[17] Gu, T. and F. Zanasi, *Functorial Semantics as a Unifying Perspective on Logic Programming*, in: F. Gadducci and A. Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, Leibniz International Proceedings in Informatics (LIPIcs) **211** (2021), pp. 17:1–17:22.

[18] Hasegawa, M., "Models of Sharing Graphs: A Categorical Semantics of let and letrec," Springer Science & Business Media, 2012.

[19] Heunen, C. and J. Vicary, "Categories for Quantum Theory: an introduction," Oxford University Press, 2019.

[20] Jaffar, J. and J.-L. Lassez, *Constraint logic programming*, in: *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1987, pp. 111–119.

[21] Kelly, G. M. and M. L. Laplaza, *Coherence for compact closed categories*, Journal of pure and applied algebra **19** (1980), pp. 193–213.

[22] Levin, L. A., *Universal sequential search problems*, Problemy peredachi informatsii **9** (1973), pp. 115–116.

[23] Lloyd, J. W., "Foundations of logic programming," Springer Science & Business Media, 2012.

[24] Piedeleu, R., "Picturing resources in concurrency," Ph.D. thesis, University of Oxford (2018).

[25] Piedeleu, R. and F. Zanasi, *A string diagrammatic axiomatisation of finite-state automata* (2020).

[26] Pipatsrisawat, K. and A. Darwiche, *On the power of clause-learning sat solvers as resolution engines*, Artificial intelligence **175** (2011), pp. 512–525.

[27] Robinson, J. A., *A machine-oriented logic based on the resolution principle*, Journal of the ACM (JACM) **12** (1965), pp. 23–41.

[28] Selinger, P., *A survey of graphical languages for monoidal categories*, in: *New structures for physics*, Springer, 2010 pp. 289–355.

[29] Zanasi, F., "Interacting Hopf Algebras-the Theory of Linear Systems," Ph.D. thesis, Ecole normale supérieure de lyon-ENS LYON (2015).

A Representing monotone functions as monotone relations

Let us first restate the definition of a monotone relation representing a monotone function.

Definition A.1 Let $\langle X, \leq_X\rangle, \langle Y, \leq_Y\rangle$ be two partially ordered sets. We say a monotone function $f\colon X \to Y$ is *represented by* a monotone relation $R \subseteq X \times Y$ if for arbitrary $x \in X$ and $y \in Y$, $(x, y) \in R$ if and only if $f(x) \leq y$.

Proposition A.2 *If two monotone functions f, g are represented by the same monotone relation R, then $f = g$.*

Proof. We prove by contradiction. Suppose $f \neq g$ and they are represented by the same relation R, then there exists $x \in X$ such that $f(x) \neq g(x)$. Yet by the definition of R, both $(x, f(x))$ and $(x, g(x))$ are in R. We claim that this entails $f(x) = g(x)$.

Since R represents g, $(x, f(x)) \in R$ implies that there exists $x' \in X$ such that $x \leq x'$ and $g(x') \leq f(x)$. Note that g is monotone, so $g(x) \leq g(x') \leq f(x)$. Similarly, R represents f and f is monotone imply that $f(x) \leq g(x)$. Now that $\langle Y, \leq_Y\rangle$ is a partial order means that $\leq_Y$ is antisymmetric, so $f(x) = g(x)$, which contradicts the assumption that $f(x) \neq g(x)$. Therefore $f = g$. □

Proposition A.3 *The notion 'being represented by' is closed under both sequential and parallel compositions:*

(i) *If $f\colon X \to Y$ and $g\colon Y \to Z$ are represented by $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, respectively, then $f\,;g\colon X \to Z$ is represented by $R \fatsemi S$.*

(ii) *If $f\colon X \to Y$ and $g\colon U \to V$ are represented by $R \subseteq X \times Y$ and $S \subseteq U \times V$, respectively, then $f \times g\colon X \times U \to Y \times V$ is represented by $R \times S \subseteq (X \times U) \times (Y \times V)$.*

Proof.

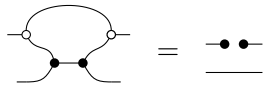
(i) We show that for arbitrary $x \in X$ and $z \in Z$, $(x, z) \in R \fatsemi S$ if and only if $(f\,;g)(x) \leq z$. For the 'only if' direction, suppose $(x, z) \in R \fatsemi S$, then there exists $y \in Y$ such that $(x, y) \in R$ and $(y, z) \in S$. Since R and S represent f and g respectively, we know that $f(x) \leq y$ and $g(y) \leq z$, so by the monotonicity of g, $(f\,;g)(x) = g(f(x)) \leq g(y) \leq z$ holds. For the 'if' direction, suppose $(f\,;g)(x) \leq z$, then $f(x) \leq f(x)$ implies that $(x, f(x)) \in R$. Also, $g(f(x)) \leq z$ implies that $(f(x), z) \in S$. So $(x, z) \in R \fatsemi S$.

(ii) This is immediate from that both $f \times g$ and $R \times S$ are defined pointwise.

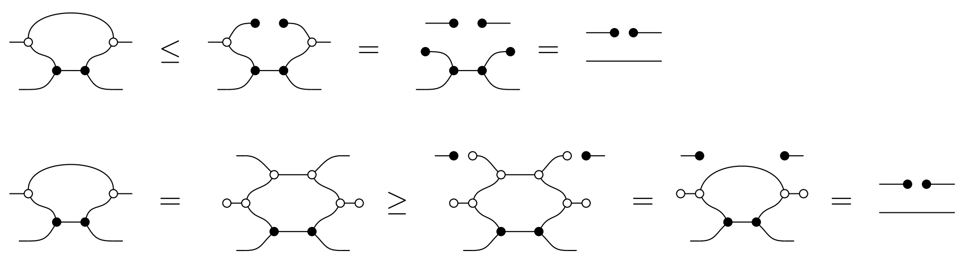
□

B Some useful diagrammatic (in)equations

Proposition B.1



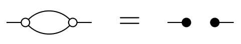
Proof. Crucially we need (A14) for the second inequality.

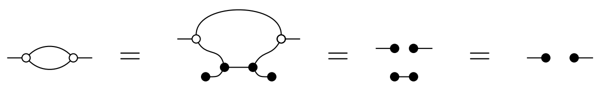


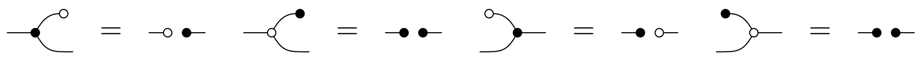
(B.1)

□

Corollary B.2

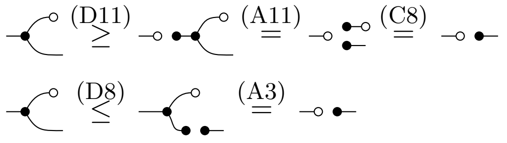


Proof.  □

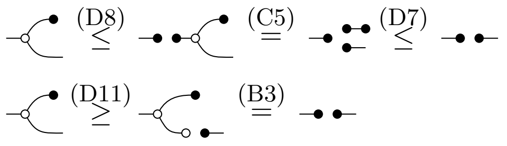
Proposition B.3 

Proof. We only prove the first two equations, and the other two follow from a similar proof by reflecting all the diagrams in the proof.

For the first equation,

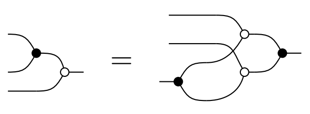


For the second equation,



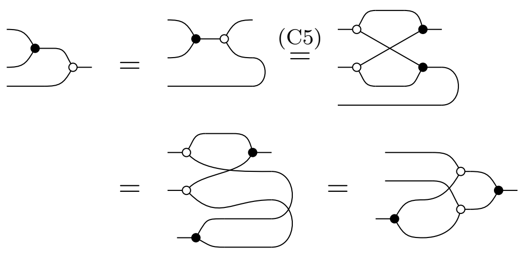
□

Proposition B.4 *The following distributivity equation holds:*



(B.2)

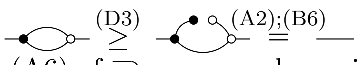
Proof.

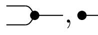


(B.3)

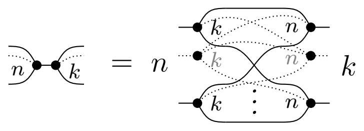
□

Proposition B.5 *The converse inequalities to (D2), (D5), and (D10) are derivable.*

Proof. For (D2) we can derive the converse inequality as follows: 

The other two are similar. For (D5) replace (D3) by (D7) and apply unitality (A6) of  and counitality (A6) of , . For (D10) replace (D3) by (D7) and apply unitality (A6) of , and counitality (B2) of , . □

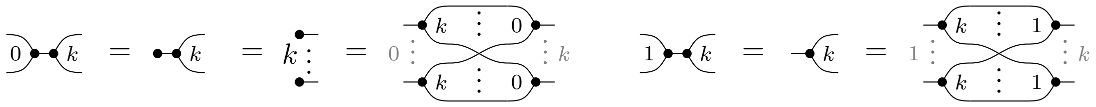
Proposition B.6 *The following general bialgebra equation holds for all natural numbers k, n.*



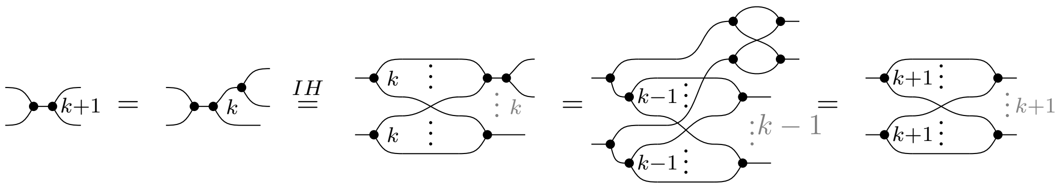
(B.4)

Proof. For readability, in the rest of the proof we will not use the dashed wires but instead use ⋮ and natural numbers to intuitively express that 'there are k duplicates of certain pattern'.

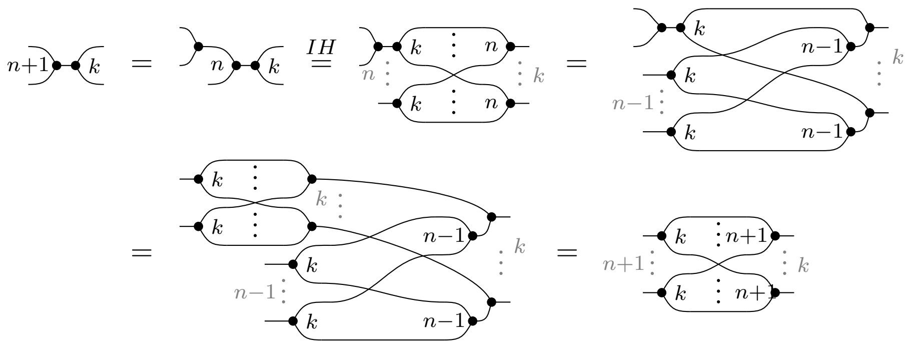
We prove by induction on n, and leave k arbitrary. When $n = 0$ and $n = 1$, we have



When $n = 2$, we make an induction on k. The base case for $k = 0, 1$ are exactly the same as the previous proof for $n = 0, 1$. For $k = 2$, this is exactly the bialgebra axiom. Now suppose it holds for $k \geq 2$, and we consider $k + 1$:

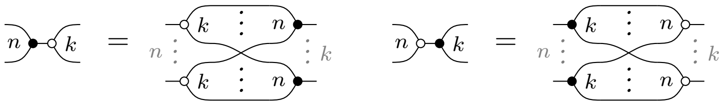


We continue with the induction proof on n. Suppose equation (B.4) holds for n, then for $n + 1$,



□

Proposition B.7 *The following bialgebra equation between and (resp. and) holds for arbitrary natural numbers k, n*

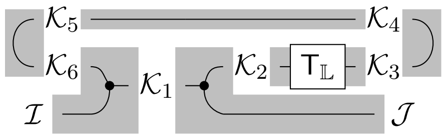


(B.5)

Proof. The proof is the same as that for Proposition B.6. Note that there only properties of (,) forms a bimonoid is used, which also holds for (,) and (,), see the B and C axioms in Figure (1). □

C Omitted proofs in Section 5

Proof of Proposition 5.4. We show that for arbitrary interpretations $\mathcal{I}, \mathcal{J}$ on At, $(\mathcal{I},\mathcal{J}) \in \langle \mathsf{C}_{\mathbb{L}} \rangle$ if and only if $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) \leq \mathcal{J}$.



(C.1)

By the definition of the interpretation $\langle \cdot \rangle$, $(\mathcal{I},\mathcal{J}) \in \langle \mathsf{C}_{\mathbb{L}} \rangle$ if and only if there exist $\mathcal{K}_i$ for $i = 1,\ldots,6$ such that: $\mathcal{I},\mathcal{K}_6 \leq \mathcal{K}_1$; $\mathcal{K}_1 \leq \mathcal{K}_2, \mathcal{J}$; $(\mathcal{K}_2,\mathcal{K}_3) \in \langle \mathsf{T}_{\mathbb{L}} \rangle$; $\mathcal{K}_3 \wedge \mathcal{K}_4 \leq \mathbf{0}$; $\mathcal{K}_5 \leq \mathcal{K}_4$; $\mathbf{1} \leq \mathcal{K}_5 \vee \mathcal{K}_6$. See (C.1) for an illustration of these $\mathcal{K}_i$. We can simplify this condition by a sequence of equivalent conditions on $\mathcal{I},\mathcal{J}$ and those $\mathcal{K}_i$:

$$\mathcal{I},\mathcal{K}_6 \leq \mathcal{K}_2,\mathcal{J};\ (\mathcal{K}_2,\mathcal{K}_3) \in \langle \mathsf{T}_{\mathbb{L}} \rangle;\ \mathcal{K}_3 \wedge \mathcal{K}_4 \leq \mathbf{0};\ \mathcal{K}_5 \leq \mathcal{K}_4;\ \mathbf{1} \leq \mathcal{K}_5 \vee \mathcal{K}_6.$$
$$\mathcal{I},\mathcal{K}_6 \leq \mathcal{K}_2,\mathcal{J};\ (\mathcal{K}_2,\mathcal{K}_3) \in \langle \mathsf{T}_{\mathbb{L}} \rangle;\ \mathcal{K}_3 \leq \mathcal{K}_6.$$
$$\mathcal{I},\mathcal{K}_3 \leq \mathcal{K}_2,\mathcal{J};\ (\mathcal{K}_2,\mathcal{K}_3) \in \langle \mathsf{T}_{\mathbb{L}} \rangle.$$
$$\mathcal{I},\mathcal{K}_3 \leq \mathcal{K}_2,\mathcal{J};\ \mathbf{T}_{\mathbb{L}}(\mathcal{K}_2) \leq \mathcal{K}_3.$$
$$\mathcal{I},\mathbf{T}_{\mathbb{L}}(\mathcal{K}_2) \leq \mathcal{K}_2,\mathcal{J}.$$

That is to say, $(\mathcal{I},\mathcal{J}) \in \langle \mathsf{C}_{\mathbb{L}} \rangle$ if and only if there exists $\mathcal{K}$ such that $\mathcal{I},\mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{K},\mathcal{J}$. We claim that this is equivalent to that $\mathcal{J} \geq \mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$, where $\mu U.F(U)$ denotes the least fixed point of a monotone function F on some complete lattice. On one hand, if $\mathcal{J} \geq \mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$, then let $\mathcal{K}$ be $\mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$. By the definition of fixed points, $\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(\mathcal{K}) = \mathcal{K}$, we know that $\mathcal{I} \leq \mathcal{K}$ and $\mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{K}$. So $\mathcal{I},\mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{J}$ as well. On the other hand, suppose there exists $\mathcal{K}$ such that $\mathcal{I},\mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{K},\mathcal{J}$, then this is equivalent to $\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{K},\mathcal{J}$. $\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(\mathcal{K}) \leq \mathcal{K}$ means that $\mathcal{K}$ is a pre-fixed point for $\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(-)\colon \{0,1\}^{At} \to \{0,1\}^{At}$, and by the Knaster-Tarski theorem, $\mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U) \leq \mathcal{K}$. Therefore, using the fact that $\mathbf{T}_{\mathbb{L}}$ is monotone, we have

$$\mathcal{J} \geq \mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(\mathcal{K}) \geq \mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(\mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)) = \mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U) \tag{C.2}$$

Now note that $\mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$ is exactly $\mathbf{C}_{\mathbb{L}}(\mathcal{I})$ (Lemma C.1), so we have $(\mathcal{I},\mathcal{J}) \in \langle \mathsf{C}_{\mathbb{L}} \rangle$ if and only if $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) \leq \mathcal{J}$, which means that $\langle \mathsf{C}_{\mathbb{L}} \rangle$ represents the consequence operator $\mathbf{C}_{\mathbb{L}}$. □

Proof of Proposition 5.4. We show that for arbitrary interpretations $\mathcal{I}$, $\mathcal{J}$ on At, $(\mathcal{I},\mathcal{J}) \in \langle \mathsf{T}_{\mathbb{L}} \rangle$ if and only if $\mathbf{T}_{\mathbb{L}}(\mathcal{I}) \leq \mathcal{J}$. We start from the left-hand-side and use the notation from Definition 5.2,

$$\begin{aligned}
&\iff \exists \mathcal{K}_1 \in \{0,1\}^{\ell_1},\ldots,\mathcal{K}_n \in \{0,1\}^{\ell_n}\colon (\mathcal{I},(\mathcal{K}_1,\ldots,\mathcal{K}_n)) \in \langle \tau_1\,;\sigma_1\,;\tau_2\,;\sigma_2\rangle, ((\mathcal{K}_1,\ldots,\mathcal{K}_n),\mathcal{J}) \in \langle \tau_3 \rangle\\
&\iff \exists \mathcal{K}_1,\ldots,\mathcal{K}_n\colon (\mathcal{I},(\mathcal{K}_1,\ldots,\mathcal{K}_n)) \in \langle \tau_1\,;\sigma_1\,;\tau_2\,;\sigma_2\rangle, \mathcal{K}_i(j_i) \leq \mathcal{J}(i) \text{ for } \forall i = 1,\ldots,n, j_i = 1,\ldots,\ell_i\\
&\quad (\mathcal{I},\mathcal{J}) \in \langle \mathsf{T}_{\mathbb{L}} \rangle\\
&\iff \text{for each } \mathbb{L}\text{-clause } \psi \equiv a_{i_1},\ldots,a_{i_r} \to a_{i_s}, \mathcal{I}(i_1) \wedge \cdots \wedge \mathcal{I}(i_r) \leq \mathcal{J}(i_s)\\
&\iff \text{for each } \mathbb{L}\text{-clause } \psi \equiv a_{i_1},\ldots,a_{i_r} \to a_{i_s}, \text{if } \mathcal{I}(i_1) = \cdots = \mathcal{I}(i_r) = 1, \text{then } \mathcal{J}(i_s) = 1\\
&\iff \mathbf{T}_{\mathbb{L}}(\mathcal{I}) \leq \mathcal{J}
\end{aligned}$$

Then by Definition A.1, this means that the monotone relation $\langle \mathsf{T}_{\mathbb{L}} \rangle$ represents the immediate consequence operator $\mathbf{T}_{\mathbb{L}}$. □

Proof of Theorem 5.6 By Theorem 3.3, $d^{\mathcal{I}}\,;\mathsf{C}_{\mathbb{L}} = d^{\mathcal{J}}$ if and only if $\left\langle d^{\mathcal{I}}\,;\mathbf{C}_{\mathbb{L}}\right\rangle = \left\langle d^{\mathcal{J}}\right\rangle$. Since $\left\langle d^{\mathcal{I}}\right\rangle$ and $\langle \mathsf{C}_{\mathbb{L}} \rangle$ represent $\mathcal{I}$ and $\mathbf{C}_{\mathbb{L}}$ respectively, $\left\langle d^{\mathcal{I}}\,;\mathsf{C}_{\mathbb{L}}\right\rangle = \left\langle d^{\mathcal{I}}\right\rangle \fatsemi \langle \mathsf{C}_{\mathbb{L}} \rangle$ represents $\mathbf{C}_{\mathbb{L}}(\mathcal{I})$, by the compositionality of representations A.3. Then $\left\langle d^{\mathcal{I}}\,;\mathsf{C}_{\mathbb{L}}\right\rangle = \left\langle d^{\mathcal{J}}\right\rangle$ is equivalent to $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mathcal{J}$. □

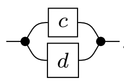
Proof of Theorem 5.7 Note that $\langle \mathsf{C}_{\mathbb{P}} \rangle$ represents $\mathbf{C}_{\mathbb{P}}$ (Proposition 5.4). By Proposition A.2, $\mathbf{C}_{\mathbb{P}_1} = \mathbf{C}_{\mathbb{P}_2}$ if and only if $\langle \mathsf{C}_{\mathbb{P}_1} \rangle = \langle \mathsf{C}_{\mathbb{P}_2} \rangle$. Then Theorem 3.3 implies that this is equivalent to $\mathsf{C}_{\mathbb{P}_1} = \mathsf{C}_{\mathbb{P}_2}$. □

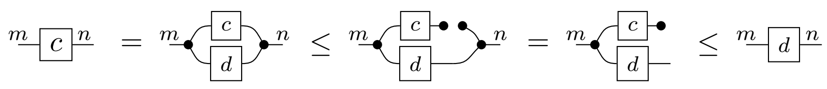
Lemma C.1 $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mu U.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$.

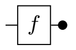
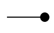
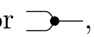
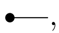
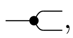
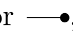
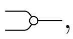
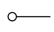
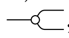
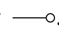
Proof. By the definition of consequence operators in Section 5, $\mathbf{C}_{\mathbb{L}}(\mathcal{I})$ is the least Herbrand model of the program $\mathbb{L} \cup \{\to a \mid a \in At, \mathcal{I}(a) = 1\}$. So $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mu U.\mathbf{T}_{\mathbb{L}\cup\mathcal{I}}(U)$. Note that for arbitrary programs $\mathbb{L}_1$ and $\mathbb{L}_2$ on At, $\mathbf{T}_{\mathbb{L}_1\cup\mathbb{L}_2} = \mathbf{T}_{\mathbb{L}_1} \vee \mathbf{T}_{\mathbb{L}_2}$, because anything that is derivable in one step from $\mathcal{I}$ using the union program $\mathbb{L}_1 \cup \mathbb{L}_2$ is either derivable via $\mathbb{L}_1$ or derivable via $\mathbb{L}_2$. Therefore $\mu U.\mathbf{T}_{\mathbb{L}\cup\mathcal{I}}(U) = \mu U.\mathbf{T}_{\mathbb{L}}(U) \vee \mathbf{T}_{\mathbb{P}_\mathcal{I}}(U)$, where $\mathbb{P}_\mathcal{I}$ is the program $\{\to a \mid a \in At, \mathcal{I}(a) = 1\}$. Since the program $\mathbb{P}_\mathcal{I}$ consists solely of facts, $\mathbf{T}_{\mathbb{P}_\mathcal{I}}$ maps any interpretation $\mathcal{J}$ to $\mathcal{I}$, so $\mathbf{C}_{\mathbb{L}}(\mathcal{I}) = \mu U.\mathbf{T}_{\mathbb{L}}(U) \vee \mathbf{T}_{\mathbb{P}_\mathcal{I}}(U) = \mu.\mathcal{I} \vee \mathbf{T}_{\mathbb{L}}(U)$. □

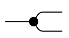
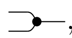
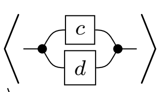
D Appendix of Section 6

D.1 Simplifying assumptions

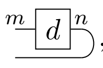
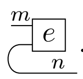
Proof of Lemma 6.1 For *(i)*, assume that $c =$ . We have

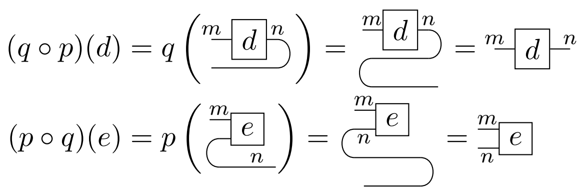


The last step uses the fact that  $\leq$  for all diagrams $f\colon m \to n$ which can proved via induction on the structure of f: the inductive cases are trivial and the base cases are given by axioms (A10) for , (A12) for , (A2)-(A3) for , trivial for , (C3) for , (C4) for , (A10) followed by (D10) for , (D8) followed by (C4) for .

For *(ii)*, assume that $\langle c\rangle \subseteq \langle d\rangle$. According to the semantics of  and , $\Big\langle$  $\Big\rangle = \{(x, y) \mid \exists x_1, x_2, y_1, y_2 \colon (x_1, y_1) \in \langle c\rangle, (x_2, y_2) \in \langle d\rangle, x \leq x_i, y_i \leq y, i = 1, 2\} = \langle c\rangle \cap \langle d\rangle = \langle c\rangle$. □

Proof of Lemma 6.2. Towards the isomorphism, we define two functions $p\colon \mathsf{Syn}[m, n] \to \mathsf{Syn}[m + n, 0]$ and $q\colon \mathsf{Syn}[m + n, 0] \to \mathsf{Syn}[m, n]$, and prove that they are inverses to each other.

Given arbitrary $d\colon m \to n$ and $e\colon m+n \to 0$, we define $p(d)$ to be , and $q(e)$ to be . Then one can show that $q \circ p$ and $p \circ q$ are identity functions, using the compact closed structure:

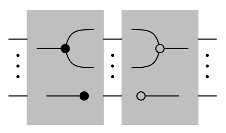


This shows that p and q witness the bijection between $\mathsf{Syn}[m, n]$ and $\mathsf{Syn}[m + n, 0]$. □

D.2 Matrices, diagrammatically

Our syntax contains a diagrammatic counterpart of Boolean matrices, about which we need to prove a few technical lemmas.

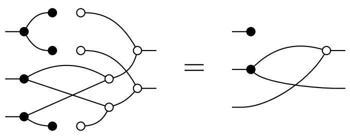
Definition D.1 We say that d is a *matrix diagram* if it factorises as follows:



(D.1)

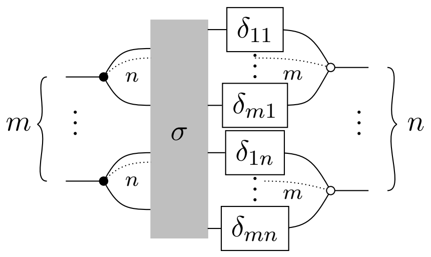
Matrix diagrams $m \to n$ denote relations that are the satisfying assignments of sets of Horn clauses, *i.e.* clauses with at most one positive (unnegated) literal. For each Horn clause, the negated literals correspond to a subset of the left wires, while the only positive literal corresponds to the right wire to which they are connected. For this reason, matrix diagrams can also be thought of simply as matrices with Boolean coefficients: each row encodes the negative literals of a Horn clause through its coefficients. An arbitrary $n \times m$ matrix A can be represented by a matrix diagram with m wires on the left and n wires on the right—the left ports can be interpreted as the columns and the right ports as the rows of A. The j-th wire on the left is connected to the i-th wire on the right whenever $A_{ij} = 1$; when $A_{ij} = 0$ they remain disconnected. For example,

the Boolean matrix $A = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ can be depicted as



Conversely, given a diagram of this form, we can recover the corresponding matrix by looking at which ports are connected to which.

Definition D.2 A *canonical matrix diagram* $d\colon m \to n$ is a diagram of the following form:



(D.2)

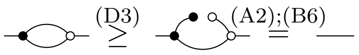
where each δ_{ij} is either or , for $i = 0, \dots, m-1$ and $j = 0, \dots, n-1$; σ contains suitably many that move the j-th output wire of the i-th to the i-th input wire of the j-th .

The equational theory *SATA* is complete for the monoidal subcategory generated by , , , .

Proposition D.3 (Matrix completeness) *Let $c, d\colon m \to n$ be two diagrams formed only of the generators , , , . We have $\langle c \rangle = \langle d \rangle$ iff $c = d$ in SATA.*

Proof. The symmetric monoidal category of matrices with Boolean coefficients and the direct sum as monoidal product has a well-known complete axiomatisation: the theory of a commutative and idempotent bimonoid. This result can be found in [29, Proposition 3.9]: therein the third author of this paper gives a complete axiomatisation of the symmetric monoidal category of matrices over a principal ideal domain (again with the direct sum as monoidal product). However the proof in [29] does not make any use of additive inverses and can be adapted without changes to the case of an arbitrary semiring, like the Booleans.

In *SATA* the axioms defining a commutative and idempotent bimonoid are given in Fig. 1 by equations (A1)-(A13). Note however that we care about matrices formed using , , , here, so we need to check that the same equations hold with replaced by and replaced by . For this, we only need to show that we can strengthen (D2) to an equality (for idempotency). The converse inequality can be derived as follows:

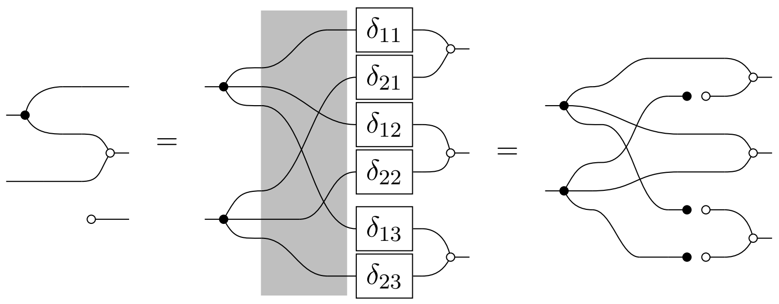


This shows that the symmetric monoidal subcategory of SATA generated by , , , embeds faithfully into the symmetric monoidal category of Boolean matrices.

The desired completeness result follows because the symmetric monoidal category of Boolean matrices embeds faithfully in our target category of monotone relations and that $\langle\cdot\rangle$ (or rather, the restriction of $\langle\cdot\rangle$ to the symmetric monoidal subcategory generated by , , ,) factors through this embedding. To see this, notice that an $n \times m$ Boolean matrix A defines a monotone function $f\colon \mathbb{B}^m \to \mathbb{B}^n$ whose j-th component is given by $f_j(x_1, \dots, x_m) = x_{j_1} \wedge \cdots \wedge x_{j_k}$ where $\{j_1, \dots, j_k\}$ is the set of indices in the j-th row of A which are equal to 1. Clearly, any two matrices that define the same monotone function are equal. In turn, we can embed monotone functions faithfully into monotone relations, as explained in Proposition A.2. □

Corollary D.4 *Any diagram formed only of the generators* —●<, —●, ○—, >○— *is equal to a canonical matrix diagram.*

Example D.5 The following matrix diagram is equivalent to a canonical matrix diagram on the right hand side:



(D.3)

In particular, the grey diagram is the σ in Definition D.2; $\delta_{21} = \delta_{13} = \delta_{23} =$ —● ○—, while the rest δ_{ij}'s are all ——.

Proposition 6.4 will allow us to simplify our reasoning in the proof of completeness. In light of this result, we will assume – wherever it is convenient – that we can always rewrite any diagram formed of —●<, —●, ○—, >○— to its equivalent matrix canonical form. This often means that we can be as lax as necessary when identifying diagrams modulo the (co)associativity of —●< or >○— and the (co)unitality of (—●<, —●) and (>○—, ○—).

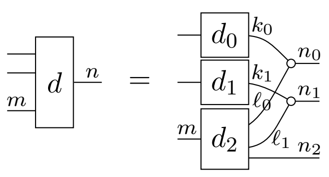
Remark D.6 The equational theory of Fig. 1 contains three commutative and idempotent bimonoids: (—●<, —●, ○—, >○—) as we have already explained, but also (—●<, —●, ●—, >●—) and (—○<, —○, ●—, >●—). Indeed, one can check that the relevant axioms also hold for them: they are easy consequences of (A1)-(A13), (C1)-(C6), and (D1)-(D12). We use the encoding of Boolean matrices using (—○<, —○, ●—, >●—) once in Section 4, when translating SAT instances to a diagrams.

Moreover, the canonical form of matrix diagrams provides a simple and formal definition of the intuitive notion of connected wires.

Definition D.7 Let d be a canonical matrix diagram as in Definition D.2. The i-th input wire and the k-th output wire are *connected* if $\delta_{ik} =$ ——. The i-th and the j-th input wires are *connected* if there exists $k \in \{0, \dots, n-1\}$ such that $\delta_{ik} = \delta_{jk} =$ ——.

Intuitively, two wires are connected if there is a path between these two wires. The notion of two wires being connected is important because when certain wires are not connected, a matrix diagram can be decomposed into smaller matrix diagrams.

Proposition D.8 *Let* $d\colon m+2 \to n$ *be a matrix diagram. If two input wires are not connected, say input wire* 0 *and* 1*, then* d *can be decomposed as follows:*

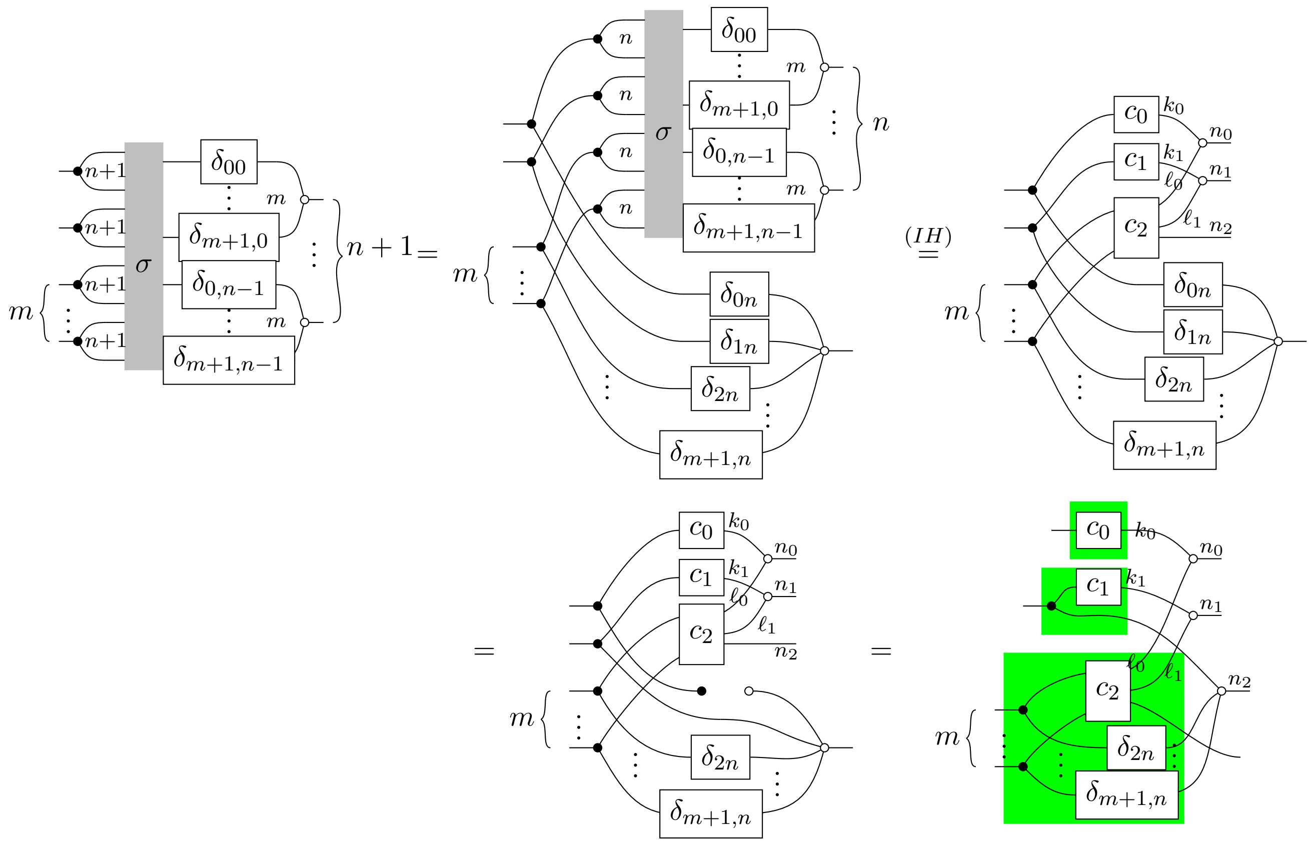


(D.4)

Proof. In this proof we will use { and natural numbers k to indicate that certain pattern appears k many times.

Without lose of generality we assume that d is a canonical matrix diagram. While one can prove directly on d for arbitrary m, n, for clearance we prove by induction on n. If $n = 0$, then (D.4) is trivial.

Suppose (D.4) works for arbitrary matrix diagram of type $m+2 \to n$, we show that it also holds for canonical matrix diagram $d\colon m+2 \to n+1$. Since input wires 0 and 1 are not connected, wlog we assume that $\delta_{0n} =$ —● ○— and $\delta_{0n} =$ —— (the case where both are —● ○— is simpler). Then d can be decomposed as follows, where the three green highlighted (matrix) diagrams work respectively as d_0, d_1, d_2 in (D.4):

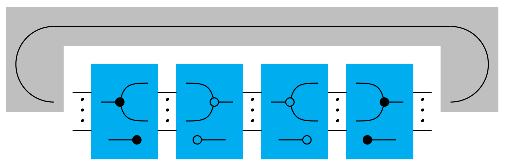


This finishes the induction step. ☐

D.3 Pre-normal form

In order to rewrite diagrams $m \to 0$ into pre-normal form, we introduce first the notion of traced canonical form for arbitrary diagrams $m \to n$.

Lemma D.9 *Every* Syn*-morphism $d\colon m \to n$ has the following* traced canonical form*:*

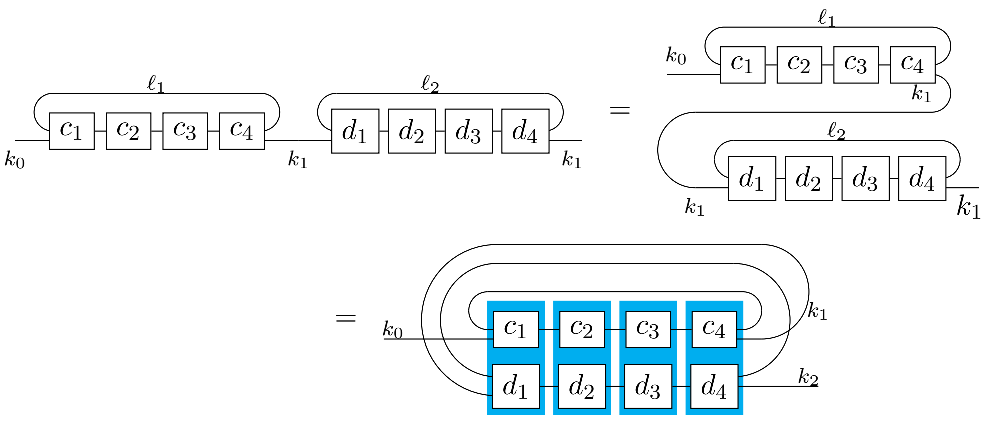


(D.5)

By bending the right wires of a diagram in traced canonical form to the left, with the mapping $\mathsf{Syn}[m,n] \to \mathsf{Syn}[0, n+m]$ provided by Lemma 6.2, we can rewrite any diagram $n \to 0$ into a form that we call *existential canonical form.*

Proof of Lemma D.9. We prove that every Syn-morphisms is equivalent to one in traced canonical form by induction on the structure of d. All the base cases (for generators) are obvious because every generator appears in (D.5). For the induction step we need to show that if c and d are two morphisms in traced canonical form, then their parallel and sequential compositions are also of traced-canonical form. For parallel composition it is also obvious. As for sequential composition, suppose the types $c\colon k_0 \to k_1$

and $d\colon k_1 \to k_2$, then $c\,;d$ is of the following form:

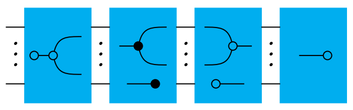


(D.6)

thus again of traced canonical form. □

Next, when we restrict ourselves to diagrams with codomain 0, we have the following existential canonical form, which are essentially traced canonical normal form diagrams with bended inputs.

Proposition D.10 *Every diagram of type $m \to 0$ is equal to one in the following* existential canonical form*:*

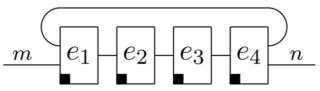


(D.7)

Notice that existential canonical form diagrams are close to being in pre-normal form: they are pre-normal form diagrams possibly pre-composed with multiple $\circ\!\!-\!\!<$. Thus, to prove that every diagram is equal to one in pre-normal form, it suffices to show how to eliminate these $\circ\!\!-\!\!<$ from existential canonical form diagrams, while keeping the pre-normal form structure unchanged.

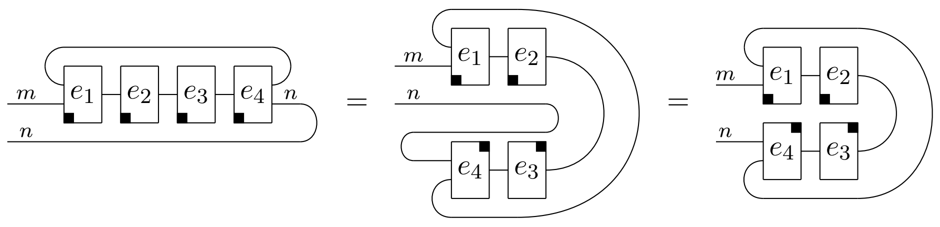
Remark D.11 The name *existential canonical form* comes from a resemblance between this form and quantified boolean formulas in prenex form, with the $\circ\!\!-\!\!<$ likened to quantifiers in this analogy. The next proof, which shows how to remove the $\circ\!\!-\!\!<$, can be thought of as the diagrammatic counterpart of a quantifier elimination procedure.

Proof of Proposition D.10. We pick arbitrary natural numbers m, n satisfying $k = m + n$. Given arbitrary morphism $d\colon k \to 0$, we apply function $q\colon \mathsf{Syn}[m+n, 0] \to \mathsf{Syn}[m, n]$ from Lemma 6.2, and get $q(d)\colon m \to n$. By Lemma D.9, $q(d)$ is of traced canonical form, say (D.8). Here each e_i is freely generated by the generators in the i-th blue box in (D.5). In this proof we also use the intuitive notation ◱ to specify the orientation of a box, which is important to keep track of when 'sliding' along the traces (see [28]).



(D.8)

Then $d = p(q(d))$ is equivalent to

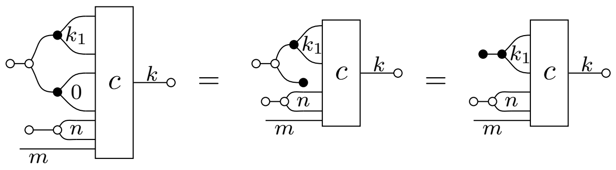


and the last diagram is of pre-normal form (D.7). □

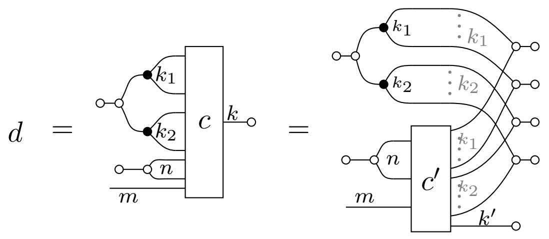
Proof of Lemma 6.3. So far we have shown that every diagram $d\colon m \to 0$ is equivalent to some existential canonical form diagram d', and such d' differs from pre-normal form only in that it may possibly be pre-composed with multiple cups (namely ○-◁⊏). So in order to show that every morphism of type $m \to 0$ is equivalent to some pre-normal form diagram, it suffices to prove that all existential canonical form diagrams are equivalent to pre-normal form diagrams by showing that one can eliminate those pre-composed ○-◁⊏ one by one.

Formally, we prove by induction on the number of ○-◁⊏ in existential canonical form diagrams. We denote the number of ○-◁⊏ in diagram d by $\#_{○-◁⊏}(d)$. Let $d\colon m \to 0$ be for existential canonical form. For the base case, if $\#_{○-◁⊏}(d) = 0$, then d is already of pre-normal form.

For the induction step, suppose $\#_{○-◁⊏}(d) = n + 1$, and we have the induction hypothesis (IH): for all existential canonical form morphisms d' with $\#_{○-◁⊏}(d') \leq n$, there exists a pre-normal form diagram d'' such that $d' = d''$. Below we eliminate the first ○-◁⊏ from the top. Without loss of generality we suppose d is of the form on the left-hand-side of (D.9), where k_1, k_2, k, n are natural numbers. In particular, we assume that this presentation 'exhausts' all —●⊏ on the right-hand-side of the top ○-◁⊏: for each wire of the $k_1 + k_2$ wires, there is no —●⊏ following it. This is always possible because c is a matrix diagram. Before moving on we solve one simple side case. If one of k_1 and k_2 is 0, then the top ○-◁⊏ can be eliminated easily. For example, if $k_2 = 0$, then



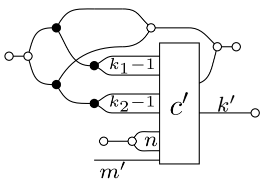
So below we assume that $k_1, k_2 \geq 1$. In this case, we claim that such d is always equivalent to a diagram on the right-hand-side below, where c' is some matrix diagram:



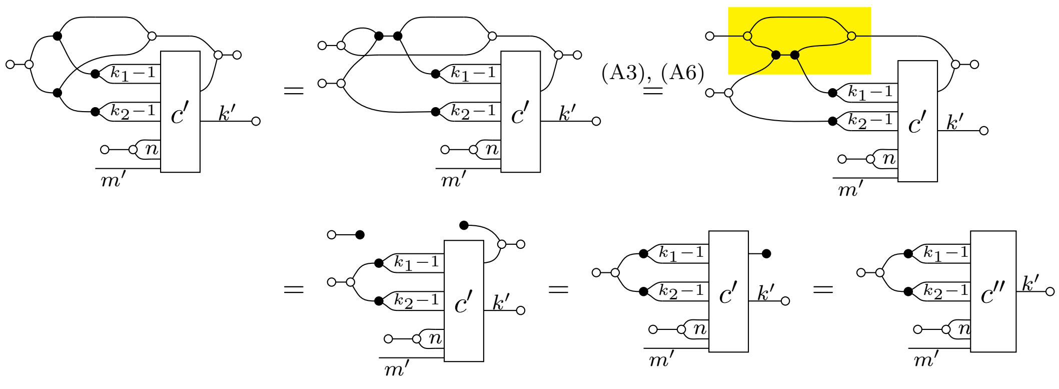
(D.9)

The key is to study whether those k_1 wires and k_2 wires are connected with each other.

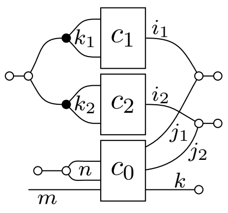
First, suppose that $k_1, k_2 \geq 1$, and that one of the k_1 wires and one of the k_2 wires are connected (as inputs to the matrix diagram c, see Definition D.7). Then d is equivalent to



This means that there is a *loop* in the diagram, which is highlighted yellow as below. As a consequence, we can 'delete' these two connected wires without affecting the other $(k_1 - 1) + (k_2 - 1)$ wires, and the resulting diagram is also of pre-normal form.



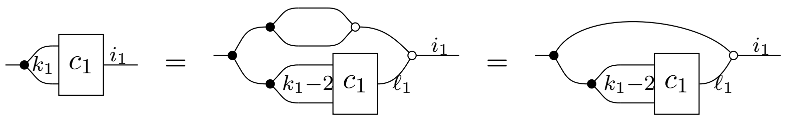
One can repeat this procedure until there is no two wires from the k_1 and k_2 wires respectively that are connected. Then by Proposition D.8, d can be decomposed as



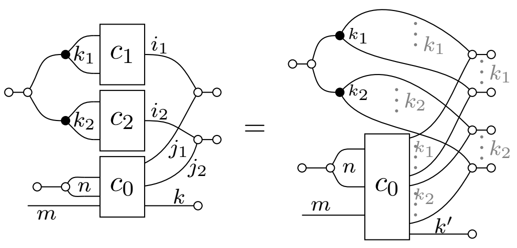
(D.10)

where c_1, c_2, c_3 are all matrix diagrams, and c_1, c_2 consist only of and (namely no nor).

Next, we consider if two of the k_i wires are connected, for $i = 1, 2$. Without loss of generality, suppose two wires among the k_1 wires are connected, then indeed these two wires are 'redundant' in the sense that the diagram is equivalent to one which has only $k_1 - 1$ copies:

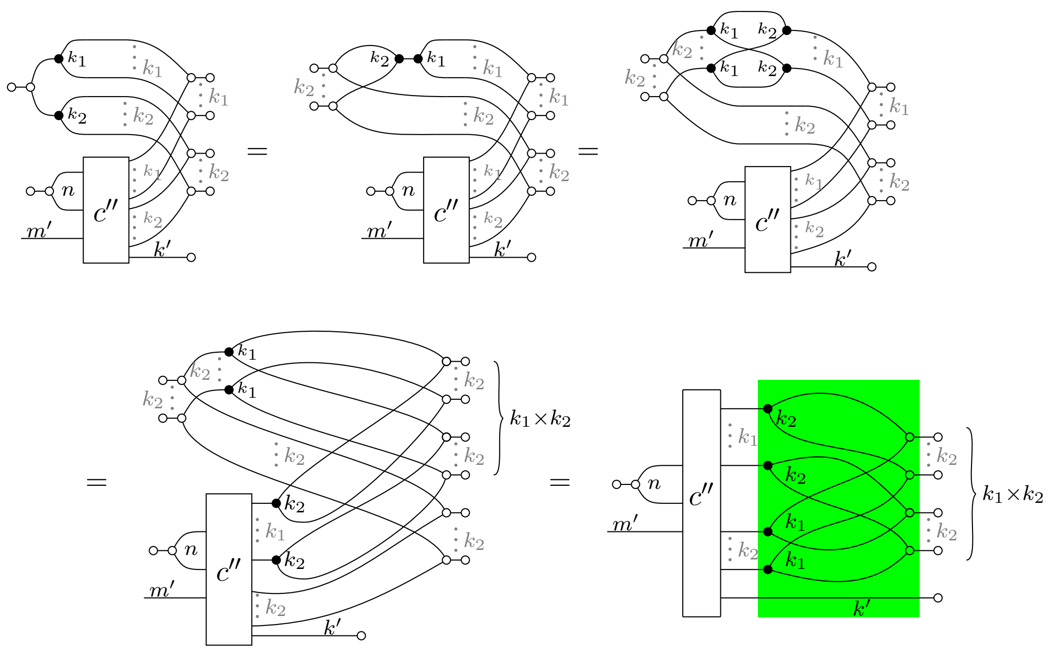


where the second step holds by Proposition B.5, which proves in particular that (D2) can be strengthened to an equality. One can repeat this procedure until either $k_i < 2$, or any two wires in the k_i wires are not connected, for $i = 1, 2$. In both cases the c_1 and c_2 boxes can be 'opened':



So far we have proved (D.9), and we are now ready to eliminate the top , using Proposition B.6

and Frobenius equations (B9), (B10):



(D.11)

Note that the last diagram in (D.11) is of existential canonical form because the green part is a matrix diagram, whose composition with the matrix diagram c'' is again equivalent to a matrix diagram, by matrix completeness (Proposition 6.4). Moreover, it now has only n-many $\multimap\!\!\subset$; thus, by the induction hypothesis it is equivalent to a pre-normal form diagram. $\square$

D.4 Downward closed subsets, sets of clauses, pre-normal form

It is helpful to first understand why every diagram with codomain 0 is equivalent to a pre-normal form diagram, from a semantic perspective. This amounts to explaining why downward closed subsets of $\mathbb{B}^m$ are (indeed, precisely) interpretations of pre-normal form diagrams by $\langle\cdot\rangle$. We first establish the connection between downward closed subsets and $\neg$-clauses, and then relate the latter with pre-normal form diagrams.

Example D.12 Let $\mathit{Var} = \{x_1, x_2, x_3\}$. The downward closed subset $A_1 = \{000, 001, 010\}$ of $\mathbb{B}^{\mathit{Var}}$ satisfies the $\neg$-clause $\{\{x_1\}, \{x_2, x_3\}\}$; the downward closed set $A_2 = \{000, 001, 010, 100, 011\}$ satisfies the $\neg$-clause $\{\{x_1, x_2\}, \{x_1, x_3\}\}$.

Lemma 6.10 together with the discussion below on pre-normal form diagrams and sets of $\neg$-clauses establish the connection between downward closed subsets and pre-normal form diagrams.

Proposition D.13 *A subset $A \subseteq \mathbb{B}^m$ is downward closed if and only if there exists some pre-normal form diagram $d\colon m \to 0$ such that $\langle d \rangle = A$.*

Proof of Proposition 6.10. The 'if' direction is straightforward. A clause $\{x_{p_1}, \ldots, x_{p_{k_i}}\}$ represents the formula $x_{p_1} \wedge \cdots \wedge x_{p_{k_i}} = 0$ or, equivalently, $\neg x_{p_1} \vee \cdots \vee \neg x_{p_{k_i}} = 1$. It is clear that the set of satisfying assignments of such a clause is downward closed, as we explained in Section 2.

For the 'only if' direction, let S be a downward closed subset of $\mathbb{B}^{\mathit{Var}}$. Let Γ be the set of all clauses whose set of satisfying assignments contain S (note that this may be the set containing the empty clause if S is empty). Then clearly, all $\mathbf{s} \in S$ satisfy Γ. Suppose that there exists some $\mathbf{t} \notin S$ which also satisfy Γ. Then write $\varphi = \{i_1, \ldots, i_p\} \subseteq \{1, \ldots, m\}$ for the set of components i_k at which there exists $\mathbf{s}_k \in S$ such that $\mathbf{s}_k(i_k) = 0 = \neg\mathbf{t}(i_k)$ (note that there is at least one such component, otherwise $\mathbf{t}$ would be in S). By construction, elements of S satisfy φ while $\mathbf{t}$ does not. This implies that φ must be in Γ, so $\mathbf{t}$ should satisfy φ too, a contradiction.

$\square$

Proof of Lemma 6.7. Note that in the construction of $\mathsf{diag}(\Phi)$, if we have that input wires $i_1, \ldots, i_k$ are connected to a ○—, then semantically it means that $\langle \mathsf{diag}(\Phi) \rangle \subseteq \{\mathbf{s} \in \mathbb{B}^{Var} \mid \mathbf{s}(i_1) \wedge \cdots \wedge \mathbf{s}(i_k) = 1\}$. In particular, if x_i does not appear in Γ, then $\langle \mathsf{diag}(\Phi) \rangle$ contains —● at the i-th input wire, and it implies the trivial condition $\langle \mathsf{diag}(\Phi) \rangle \subseteq \mathbb{B}^{Var}$. But then $\langle \mathsf{diag}(\Phi) \rangle$ is the intersection of all these sets for each $\gamma \in \Gamma$, which by definition is exactly the set of satisfying assignments of Γ. □

Lemma D.14 *Consider a* $\neg$*-clause* $\Gamma = \left\{\{y_1^1, \ldots, y_{k_1}^1\}, \ldots, \{y_1^m, \ldots, y_{k_m}^m\}\right\}$ *and a single* $\neg$*-clause* $\varphi = \{x_1, \ldots, x_n\}$*. If every satisfying assignment of* Γ *also satisfies* φ*, then there exists some* $i \in \{1, \ldots, m\}$ *such that* $\{y_1^i, \ldots, y_{k_i}^i\} \subseteq \{x_1, \ldots, x_n\}$*.*

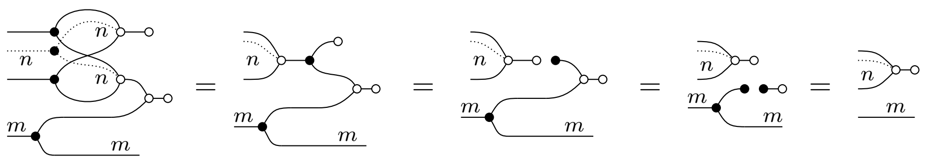
Proof. We prove this by contradiction. Consider a set of clauses $\Gamma = \left\{\{y_1^1, \ldots, y_{k_1}^1\}, \ldots, \{y_1^m, \ldots, y_{k_m}^m\}\right\}$ and the clause $\varphi = \{x_1, \ldots, x_n\}$ such that every satisfying assignment of Γ also satisfies φ. Suppose the lemma does not hold, so that for each $i \in \{1, \ldots, m\}$, there exists some $y_{p_i}^i$ such that $y_{p_i}^i \notin \{x_1, \ldots, x_n\}$. Consider the assignment $\mathbf{s}$ such that $\mathbf{s}(y_{p_i}^i) = 0$ for all $i \in \{1, \ldots, m\}$, and $\mathbf{s}(z) = 1$ for all the other variables. Then it satisfies Γ: for each i, $\mathbf{s}(y_1^i) \wedge \cdots \wedge \mathbf{s}(y_{p_i}^i) \wedge \cdots \mathbf{s}(y_{k_i}^i) = 0$. However, $\mathbf{s}$ does not satisfy φ because $\mathbf{s}(x_j) = 1$ for all $j \in \{1, \ldots, n\}$. This is a contradiction. □

Proof of Lemma 6.9. Since the satisfying assignments of Σ form a subset of those of Γ, they are also a subset of the satisfying assignments of every $\gamma \in \Gamma$. By Lemma D.14, for every $\gamma \in \Gamma$ there exists a $\sigma \in \Sigma$ such that $\sigma \subseteq \gamma$. Using a similar argument, since the satisfying assignments of Γ form a subset of those of Σ, for every $\sigma \in \Sigma$ there exists a $\gamma \in \Gamma$ such that $\gamma \subseteq \sigma$.

Now starting with an arbitrary $\gamma \in \Gamma$, there exists some $\sigma \in \Sigma$ such that $\sigma \subseteq \gamma$, and for this γ there again exists some $\sigma' \in \Sigma$ such that $\sigma' \subseteq \gamma$. So $\gamma \subseteq \gamma'$, and for a minimal set of clauses this means $\gamma \subseteq \gamma'$: otherwise we could drop γ' and get a smaller set than Γ with the same satisfying assignments. Then $\gamma \subseteq \sigma \subseteq \gamma'$ and $\gamma = \gamma'$ imply $\gamma = \sigma = \gamma'$, which means that γ is in Σ as well. Since this holds for arbitrary $\gamma \in \Gamma$, we know that $\Gamma \subseteq \Sigma$. Applying a similar argument for Σ, we know that $\Sigma \subseteq \Gamma$. Therefore we can conclude that $\Sigma = \Gamma$. □

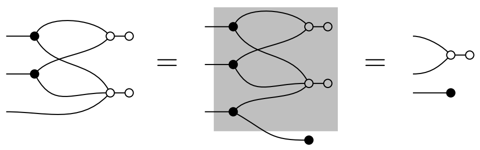
Proof of Proposition 6.14 (i) $\simeq$ (ii): It suffices to define two functions between (i) and (ii) which are injective and inverses to each other. We already know that a subset of $\mathbb{B}^m$ is downward closed if and only if it is the set of satisfying assignments to some set of $\neg$-clauses (Lemma 6.10). Starting from a downward closed subset $A \subseteq \mathbb{B}^m$, one have a least set of $\neg$-clauses whose set of satisfying assignments is A (Lemma 6.9). Inversely, every set of $\neg$-clauses uniquely determines a downward closed subset of $\mathbb{B}^m$, namely the set of its satisfying assignments. It follows immediately that these two functions are inverses to each other.
(ii) $\simeq$ (iii): We notice that when restricted to minimal sets of $\neg$-clauses (resp. normal form diagrams), the images of $\mathsf{diag}(\cdot)$ (resp. $\mathsf{clause}(\cdot)$) are normal form diagrams (resp. minimal sets of $\neg$-clause). Moreover, two normal form diagrams have the same $\mathsf{diag}(\cdot)$-image precisely when they are equivalent up to commutativity. Thus $\mathsf{diag}(\cdot)$ and $\mathsf{clause}(\cdot)$ witness the isomorphism. □

Proof of Lemma 6.16



Here the first step uses Proposition B.7; the second and third steps use Proposition B.3. □

Example D.15 Consider the pre-normal form diagram c_1 from Example 6.13, one can apply Lemma 6.16 (to the grey block below) and get a normal form diagram:



Proof of Lemma 6.18. Given a pre-normal form diagram $d\colon m \to 0$ which is *not* in normal form. By Definition 6.12, this means that in the $n \times m$ Boolean matrix A represented by d, there exist two rows i and j such that $A_{ik} \leq A_{jk}$ for all $k = 1, \ldots, m$. Diagrammatically, this means that there is the same pattern as on the lhs of Lemma 6.16: there exists some $\circ\!\!-\!\!-$ (the i-th one) such that all the input wires that are connected with this $\circ\!\!-\!\!-$ is also connected with some other $\circ\!\!-\!\!-$. In this case, we apply Lemma 6.3 (from left to right) and turn diagram d into a pre-normal form diagram e, whose representing matrix B is of type $(n-1) \times m$, and is essentially A but without the i-th row.

Since A is finite, such pattern disobeying normal form can only happen finitely many times, each can be solved by the above procedure. The resulting diagram is of normal form and equivalent to the original diagram d.

□

Proof of Lemma 6.19. As before, given Lemma 6.2 it suffices to prove this statement for morphisms with codomains 0. Let $c, d\colon m \to 0$ be two diagrams satisfying $\langle c \rangle = \langle d \rangle = A$, for some a downward closed subset A of $\mathbb{B}^{Var}$ where $Var = \{x_1, \ldots, x_m\}$. By Lemma 6.11, A is the set of satisfying assignments of some minimal set of clauses, say Γ. By Lemma 6.18, c and d are equivalent to some normal form diagrams, say $\hat{c}$ and $\hat{d}$, respectively. Moreover, $\hat{c}$ and $\hat{d}$ verify $\mathsf{clause}(\hat{c}) = \mathsf{clause}(\hat{d}) = \Gamma$.

Note that every set of clauses corresponds to a diagram in pre-normal form; when this set of clauses is minimal, the corresponding diagram is in normal form, by Proposition 6.14. Now $\hat{c}$ and $\hat{d}$ are both normal form diagrams with semantics A, so they are both the normal form diagram uniquely determined by Γ, the minimal set of clauses with satisfying assignments A. This means that $\hat{c} = \hat{d}$. □

D.5 From diagrams to clauses and back

In Subsection 6.1 we explored how to derive a unique set of $\neg$-clauses from a given diagram with the same semantics and vice versa. Both directions relies on matrix completeness (Proposition 6.4). Here instead we state how $\mathsf{clause}(\cdot)$ and $\mathsf{diag}(\cdot)$ work directly, without using matrix diagrams and their representing matrices as an intermediate step.

On one hand, let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a set of $\neg$-clauses over $Var = \{x_1, \ldots, x_m\}$, and we define $\mathsf{diag}(\Phi)$ as follows. For every $\neg$-clause $\varphi \in \Phi$ of the form $\varphi = \{x_{i_1}, \ldots, x_{i_k}\}$, one creates a copy (via $-\!\!\bullet\!\!\!\!\subset$) of each of the input wires corresponding to the variables $x_{i_1}, \ldots, x_{i_k}$, and connect them via $\overset{k}{\succ}\!\!\circ\!\!-\!\!\circ$. In particular, if a variable x_i does not appear in Φ, then we make 0 copy of the i-th input wire by post-composing it with $-\!\!-\!\!\bullet$, effectively discarding it.

On the other hand, let $d\colon m \to 0$ be a pre-normal form. Moreover, we assume that d is decomposed as $d = d_1 \,;\, d_2$, where $d_1\colon m \to n$ is a matrix diagram, and $d_2 = (\circ\!\!-\!\!-)^n$. Then we define $\mathsf{clause}(d)$ as the following set of $\neg$-clauses consisting of n $\neg$-clauses over $Var = \{x_1, \ldots, x_m\}$. Each $\circ\!\!-\!\!-$ in d_2 (thus in d) generates on $\neg$-clause φ: if $i_1, \ldots, i_k$ are all the input wires that are connected with this $\circ\!\!-\!\!-$, then $\varphi = \{x_{i_1}, \ldots, x_{i_k}\}$. Then $\mathsf{clause}(d)$ is defined as the set of all the $\neg$-clauses generated by $\circ\!\!-\!\!-$ in d. In particular, if no input wire is connected with this $-\!\!-\!\!\circ$, then it must be of the form $\circ\!\!-\!\!-\!\!\circ$, which generates the empty clause (which has an empty set of satisfying assignments).