

Categorical E-Graphs for Lambda Calculi

Aleksei Tiurin
University of Birmingham

Dan R. Ghica
University of Birmingham
and
Huawei Central Software Institute
*

Nick Hu
Kyoto University

Equality saturation is a powerful technique for program optimisation and reasoning, driven by the use of equivalence classes of terms under rewrite rules. These equivalence classes lie at the root of data structures called equality graphs (e-graphs). Despite their numerous advantages, until recently e-graphs lacked native support for variable binding, limiting their applicability to programming languages. We propose to address this problem from a categorical perspective by extending the interpretation of e-graphs as string diagrams, namely morphisms in symmetric monoidal semilattice-enriched categories which we additionally equip with a monoidal closed structure. We further define a concrete representation using hierarchical hypergraphs, and introduce a corresponding double-pushout (DPO) rewriting system. Finally, we establish the equivalence between term rewriting and DPO rewriting, with the combinatorial model inherently absorbing the equations of the symmetric monoidal structure. Our approach, specifically designed for lambda calculi, is compared and contrasted with slotted e-graphs — an alternative method for incorporating variable binding within e-graphs.

1 Introduction

1.1 Motivation and contribution

E-graphs (short for *equality graphs*) [34] are an efficient data structure for non-destructive rewriting, incorporating equivalence representation, used in automated theorem proving, compiler optimisation, and program analysis. They mitigate the *phase-ordering problem* of term rewriting: the way rewrites are scheduled may enable or block further rewrites due to introduction or elimination of redexes, resulting in a suboptimal result. This is achieved by making rewrites on e-graphs non-destructive: an application of a rewrite rule, instead of replacing the left-hand side by the right-hand side, adds the latter to the e-graph so that all pre-existing redexes are protected and new ones are introduced. Exponential growth of the e-graph is avoided by the clever use of sharing of nodes.

E-graphs were first introduced in the automated theorem proving literature in the 1980s [26], but recent years have seen renewed interest, particularly for optimisation and synthesis using the technique of *equality saturation* [32, 15, 34], wherein rewrites to an e-graph are repeated until a fixpoint (or timeout) is reached. We are interested in using e-graphs in the context of functional programming languages, which builds upon an internal representation of the λ -calculus. Köehler et al. [23] pioneered work in this area by identifying two key, interconnected challenges—handling binding and ensuring capture-avoiding substitution—when applying equality saturation to an encoding of the λ -calculus in e-graphs.

In the textual representation of λ -terms, α -equivalent terms may be distinct and thus cannot be shared. De Bruijn indices [13] syntactically identify α -equivalent terms, but the same variable can receive different indices depending on its scope, again interfering with sharing. As a simple example, $\lambda f.f((\lambda x.fx)2)$ is encoded as $\lambda.\%0((\lambda.\%1\%0)2)$, where the two occurrences of f are represented differently and cannot be shared.

*This work was supported by EPSRC grants EP/V001612/1 and EP/Y010035/1.

A simple but theoretically [1] and practically [28] important extension of the λ -calculus consists of adding let-bindings or explicit substitutions to the syntax. Terms are identified up to permutations of non-interacting substitutions, called “graph equivalence” [4]. Modelling this in conventional e-graphs requires additional nodes for explicit substitutions and additional rewrite rules to capture graphical equivalence, polluting the equality saturation state space with bureaucracy rather than the domain-specific rewriting one actually cares about.

All these issues can be resolved by the use of *string diagrams* in place of textual syntax. Tiurin et al. [33] shows how e-graphs (without binding) can be given a categorical semantics using symmetric monoidal semilattice-enriched categories, which allows e-graphs to be reconstructed as string diagrams. In this paper, we extend this technique to e-graphs with bindings, lifting to monoidal *closed* categories.

1.2 String diagrams

String diagrams provide a topological calculus for monoidal categories, representing objects as *strings* and morphisms as *nodes*. They bridge category theory—which gives semantic models to programming languages and logics—to the concrete language of graphs fundamental in compiler design; see Ghica and Zanasi [18] for the λ -calculus setting and Tiurin et al. [33] for e-graphs without bindings. In our string diagrams, which are oriented left-to-right and bottom-to-top, the structure of symmetric monoidal categories is absorbed into the graphical syntax, with composition and tensoring represented by juxtaposition and symmetries as weaving of strings.

We require a concrete encoding of string diagrams, analogous to the de Bruijn index representation of λ -terms. For this we use *hypergraphs* [6]: hypergraph isomorphism rigorously expresses when string diagrams “look the same” topologically, and string-diagrammatic equational reasoning is formalised as DPO rewriting over these hypergraphs [6]. For variable binding and equivalence classes we rely on *functorial boxes* [24], represented via *hierarchical hypergraphs* [5].

Revisiting the earlier example in Figure 1, observe that the two occurrences of f are now shared. The scope of a λ -abstraction is a rounded box with the bound variable as a wire on the frame, the application node a half-circle $\frown$, and sharing a *copy node* Ψ .

1.3 E-graphs

E-graphs simultaneously represent several equivalent terms of an algebraic theory by generalising *term graphs* to include equivalence classes of subterms. *E-classes* (dashed boxes) contain *e-nodes* representing equivalent terms with the same root; e-nodes reference e-classes rather than other e-nodes. Extracting a term amounts to choosing one e-node per e-class; different choices yield equivalent terms. Figure 2 shows how a series of rewrite rules are non-destructively applied to an e-graph [34], alongside the equivalent string diagram representation using *e-hypergraphs* [33]. Applying a rewrite rule to an e-graph amounts to adding new e-nodes into the e-graph and *merging* e-classes for the left-hand side and right-hand side of the rewrite rule, instantiating the variables in the rule with the e-classes according to the match. The key difference that can be noted in string-diagrammatic representation, is that the dashed boxes that correspond to e-graph e-classes are just ordinary nodes that happen to have hierarchical structure. These boxes have enough inputs to cater for each subdiagram inside, explicitly deleting the unnecessary wires inside each component using $\dagger$.

1.4 Conventional vs. slotted vs. categorical e-graphs with binding

A common approach to β -reduction in e-graphs uses special explicit-substitution nodes with associated rewrite rules [34, 23]. In our categorical representation these nodes are unnecessary, since explicit

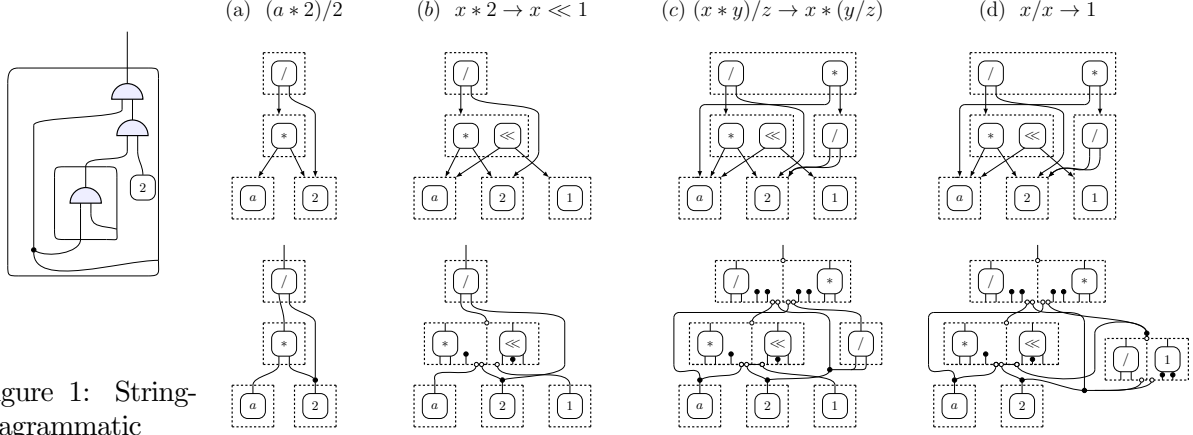
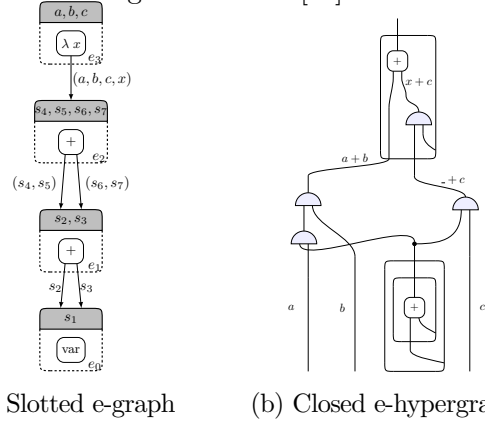


Figure 1: String-diagrammatic representation of $\lambda f.f((\lambda x.f x)2)$.

Figure 2: E-graph example (top) and its equivalent string diagram representation (bottom) [33]

substitution is modelled as composition. Figure 4a illustrates successive β -reductions in a conventional e-graph for $(\lambda x.(y+y)+x)1$; Figure 4b shows the same reduction via string diagrams, where β -reduction is simply re-wiring when an application node meets an abstraction. Using de Bruijn indices instead would only push the problem elsewhere: it greatly increases the complexity of encoding the reduction as a rewrite rule, requiring meta-syntactic shifting operators on the right-hand side [23].

Recent work on *slotted* e-graphs [29] addresses the sharing issues by incorporating variables as first-class data and making binding a built-in operation, guaranteeing freshness inside binders and differentiating bound from free variables. An example is shown in Figure 3a: each e-class is parameterised by a set of free variables and can be “invoked” by supplying concrete variable names. We can achieve a similar degree of sharing via closure conversion in our setting (Figure 3b). The key difference is that slotted e-graphs solve variable binding generically using nominal techniques applicable to arbitrary formal languages with binders (e.g. π -calculus), while our approach embeds into the internal language of our categorical model, using metatheoretic λ -abstraction to handle variable binding. We provide a case study in Section 7, where we argue that our approach has particular advantages for e-graphs for (variations of) λ -calculus. Our categorical model can also help justify the congruences maintained by slotted e-graphs, as we elaborate in Section 2. We note that Schneider et al. [29] focus on the e-graph as a data structure with a full implementation, while we focus on the foundational categorical semantics of e-graphs with binding and their combinatorial representation. We hope our work serves as a foundation for semantics of e-graphs with bindings, a currently underdeveloped area [35, 31].



(a) Slotted e-graph (b) Closed e-hypergraph
Figure 3: Extensions of e-graphs to handle binding in $\lambda x.(a+b)+(x+c)$.

1.5 Contributions

Our contribution is the construction of a form of e-graphs that support binding, developed through a principled categorical approach. Specifically, we extend a prior categorical semantics for e-graphs

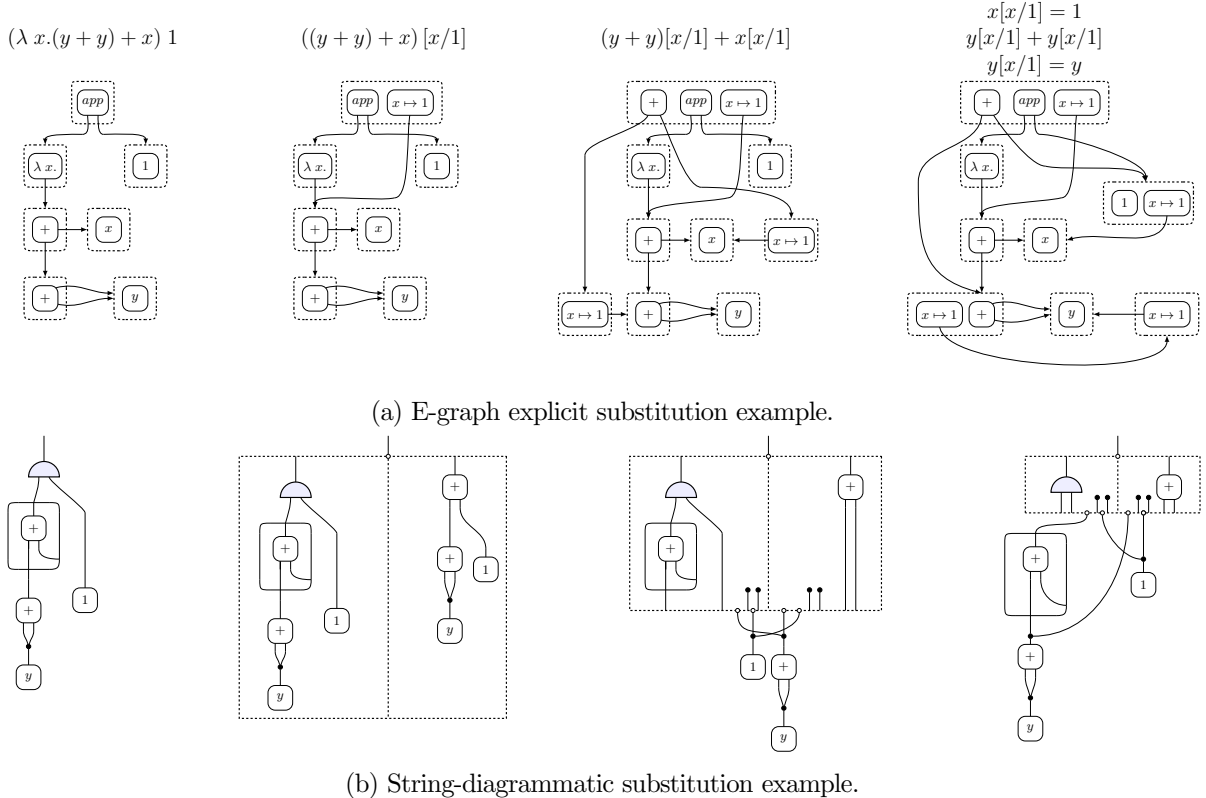


Figure 4: Conventional e-graphs vs. categorical e-graphs with binding.

without binding by incorporating a monoidal closed structure. Tiurin et al. [33] shows that e-graphs over signature Σ can be represented as morphisms of a free Cartesian symmetric monoidal semilattice-enriched category over the same signature. We argue that restricting the categorical domain to free **closed** symmetric monoidal semilattice-enriched categories gives rise to structures that naturally support binding and equivalence classes of morphisms. That is, morphisms (string diagrams) of such category can both encode variable binding (using the closed structure) and equivalence (using the semilattice enrichment) between subterms (sub-diagrams). We call these string diagrams, or more precisely, the combinatorial representation of them, *closed e-hypergraphs*. First, we define closed enriched Σ -terms which generate a theory with equivalences and bindings. Then we define what closed e-hypergraphs are and provide the interpretation of closed enriched Σ -terms as cospans of e-hypergraphs (and by doing this we also formalise the corresponding string diagrams as concrete combinatorial objects). Next we formulate the DPO rewriting for closed e-hypergraphs and show that this notion of rewriting is sound and complete with respect to rewriting of closed enriched Σ -terms modulo SMC equations. We describe what benefits the closed e-hypergraph representation may bring for equality saturation in the context of λ -calculus with explicit substitution compared to conventional (slotted) e-graphs in Section 7.

2 Categorical semantics of E-Graphs with Bindings

This section introduces preliminaries on symmetric monoidal semilattice-enriched categories generated by monoidal theories, and the string diagram formalism used to represent them. We also show a specific case of closed monoidal semilattice-enriched categories and how the closed structure interacts with the enrichment.

We use $(\circ)$ to denote the composition in diagrammatic order and use sym for the symmetry natural

transformation for a (strict) tensor ($\otimes$). We also omit subscripts on identities and natural transformations where it can be inferred.

Definition 2.1. A monoidal signature $\Sigma = (\Sigma_O, \Sigma_M, t)$ is given by a set of objects (called types) Σ_O , a set of generators (operations) Σ_M and a type assignment function $t: \Sigma_M \rightarrow \Sigma_O^* \times \Sigma_O^*$ assigning lists of input and output types to a given operation.

Definition 2.2. The set of Σ -terms generated by monoidal signature Σ is given by the grammar: $\tau ::= \phi \mid \text{id}_I \mid \text{id}_A \mid \text{sym}_{A,B} \mid \tau_1 \circ \tau_2 \mid \tau_1 \otimes \tau_2$, where $\phi \in \Sigma_M$ and $A, B \in \Sigma_O$.

Identities and symmetries of complex types, e.g. $\text{id}_{A \otimes B}$ and $\text{sym}_{A \otimes B, C}$, are given by the composition and tensoring of id_A and $\text{sym}_{A,B}$ as, e.g. $\text{id}_{A \otimes B} = \text{id}_A \otimes \text{id}_B$ and $\text{sym}_{A \otimes B, C} = (\text{id}_A \otimes \text{sym}_{B,C}) \circ (\text{sym}_{A,C} \otimes \text{id}_B)$.

Definition 2.3. The free symmetric monoidal category $\mathbf{S}(\Sigma)$ is given by a set of Σ -terms quotiented by the axioms of a symmetric monoidal category. The objects are lists of types from Σ_O with tensoring given by list concatenation.

Definition 2.4. A symmetric monoidal theory $\mathbf{SMT}(\Sigma, \mathcal{E})$ is given by a set of Σ -terms quotiented by a set of equations $\mathcal{E}$ consisting of pairs of appropriately typed Σ terms $l = r$.

Definition 2.5. A categorical presentation of $\mathbf{SMT}(\Sigma, \mathcal{E})$ is a free symmetric monoidal category $\mathbf{S}(\Sigma, \mathcal{E})$ given by a set of Σ terms quotiented by the laws of a symmetric monoidal category and $\mathcal{E}$.

We can similarly consider *closed* terms that give rise to a free *closed* symmetric monoidal category.

Definition 2.6. A (right) closed monoidal category is a monoidal category $\mathbf{C}$ satisfying that for every pair of objects B, C there is an object $B \multimap C$ and a morphism $\text{ev}_{B,C}: (B \multimap C) \otimes B \rightarrow C$, and for every triple of objects A, B, C there is an operation $\Lambda_{A,B,C}: \mathbf{C}(A \otimes B, C) \rightarrow \mathbf{C}(A, B \multimap C)$ such that for all $f: A \otimes B \rightarrow C$ and $g: Z \rightarrow A$ the following equations hold:

$$\begin{aligned} f &= \Lambda_{A,B,C}(f) \otimes \text{id}_B \circ \text{ev}_{B,C}, \quad \text{id}_{B \multimap C} = \Lambda_{B \multimap C, B, C}(\text{ev}_{B,C}), \\ \Lambda_{Z,B,C}(g \otimes \text{id}_B \circ f) &= g \circ \Lambda_{A,B,C}(f). \end{aligned}$$

This is equivalent to defining a (right) closed monoidal category as one admitting an adjunction $- \otimes B \dashv B \multimap -$ for every object B . If the tensor is symmetric, then the monoidal category is *closed*.

Definition 2.7. Closed Σ -terms are defined inductively by first defining a family of objects obj_{Σ_O} .

- A designated I is in obj_{Σ_O} and for all $A \in \Sigma_O$, $A \in \text{obj}_{\Sigma_O}$.
- If $A \in \text{obj}_{\Sigma_O}$ and $B \in \text{obj}_{\Sigma_O}$, then $A \otimes B \in \text{obj}_{\Sigma_O}$ and $A \multimap B \in \text{obj}_{\Sigma_O}$.

A family of closed Σ terms is then defined by:

- $\phi: A \rightarrow B \in \Sigma_M$ is a closed Σ -term, once we turned lists into tensors;
- if A and B are obj_{Σ_O} , then id_A and $\text{sym}_{A,B}$ are closed Σ -terms;
- if $f: A \rightarrow B$ and $g: B \rightarrow C$ are closed Σ -terms, then so is $f \circ g: A \rightarrow C$;
- if $f: A \rightarrow B$ and $g: C \rightarrow D$ are closed Σ -terms, then so is $f \otimes g: A \otimes C \rightarrow B \otimes D$;
- if A and B are obj_{Σ_O} , then $\text{ev}_{A,B}: (A \multimap B \otimes A) \rightarrow B$ is a closed Σ -term;
- if $h: X \otimes A \rightarrow B$ is a closed Σ -term, then $\Lambda_{X,A,B}(h): X \rightarrow (A \multimap B)$ is a closed Σ -term.

Definition 2.8. The free closed symmetric monoidal category $\mathbf{CS}(\Sigma)$ is given by a set of closed Σ -terms quotiented by the laws of a symmetric monoidal category and the equations from Definition 2.6. We can similarly consider a free closed symmetric monoidal category $\mathbf{CS}(\Sigma, \mathcal{E})$ induced by a $\mathbf{SMT}(\Sigma, \mathcal{E})$ over closed Σ -terms.

To encode equivalences between morphisms (or Σ -terms), the Hom sets of the corresponding free categories can be endowed with the additional structure of a semilattice induced by an idempotent, commutative and associative operator *join* $+$ as presented by Tiurin et al. [33]. This is modelled by enrichment in semilattices, yielding **SLat**-categories where **SLat** is the category of *unbounded* semilattices and homomorphisms between them (see Appendix A).

Definition 2.9 (SLat-category). An **SLat**-category $\mathbb{C}$ is defined by the following data:

- a set of objects $\mathbb{C}$;
- for every pair of objects $A, B \in \mathbb{C}$ — a Hom semilattice $\mathbb{C}(A, B) \in \mathbf{SLat}$;

- for every triple of objects $A, B, C \in \mathbb{C}$ — a composition morphism $\circ: \mathbb{C}(B, C) \otimes \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$ of **SLat**;
- for every object $A \in \mathbb{C}$ — a unit morphism $u_A: I \rightarrow \mathbb{C}(A, A)$, where I is the monoidal unit $\{*\}$ for **SLat**¹,

such that the coherence diagrams for associativity and unitality of composition commute.

SLat is sufficiently close to **Set** that Hom semilattices can be thought of as the ordinary notion of Hom sets with additional semilattice structure; unit morphisms are as usual, and composition works similarly to composition in categories with an extra condition of respecting the join: as it is a semilattice homomorphism, we require that $(f+g) \circ h = f \circ h + g \circ h$ and $f \circ (g+h) = f \circ g + f \circ h$.

Definition 2.10 (**SLat**-functor). Let $\mathbb{C}$ and $\mathbb{D}$ be two **SLat**-categories. An **SLat**-functor $F: \mathbb{C} \rightarrow \mathbb{D}$ is defined by the following data:

- a mapping on objects A of $\mathbb{C}$ to FA of $\mathbb{D}$;
- for each pair of objects A, B of $\mathbb{C}$, a morphism in **SLat** $F_{A,B}: \mathbb{C}(A, B) \rightarrow \mathbb{D}(FA, FB)$;

such that certain coherence diagrams, enforcing the preservation of composition and units, commute.

As before, this is like an ordinary functor, except with an additional requirement that its action on morphisms respects the join: $F(f+g) = F(f) + F(g)$.

The collection of **SLat**-categories and **SLat**-functors form an **SLat**-category **SLat-Cat**², and for any pair of **SLat**-categories $\mathbb{C}$ and $\mathbb{D}$, the product **SLat**-category $\mathbb{C} \times \mathbb{D}$ is defined analogously to categories.

Definition 2.11 (Symmetric monoidal **SLat**-category). A symmetric monoidal **SLat**-category is given by a **SLat**-category $\mathbb{C}$ that additionally has: a unit object $I \in \mathbb{C}$, an **SLat**-functor tensor $\otimes: \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$, such that particular coherence diagrams analogous to those for symmetric monoidal categories commute.

This is completely analogous to a symmetric monoidal category, with the fact that the tensor is an **SLat**-functor ensuring compatibility with the join: $f \otimes (g+h) = f \otimes g + f \otimes h$ and $(f+g) \otimes h = f \otimes h + g \otimes h$. We take $\otimes$ to bind more tightly than $+$.

Proposition 2.12. (A specialised case of Proposition 6.4.7 [8] and Theorem 5.7.1 [12]) There is a 2-adjunction $\mathcal{F} \dashv \mathcal{U}: \mathbf{SLat-Cat} \rightarrow \mathbf{Cat}$ induced by the usual (symmetric monoidal) free-forgetful adjunction $F \dashv U: \mathbf{SLat} \rightarrow \mathbf{Set}$. Moreover, $\mathcal{F}$ and $\mathcal{U}$ preserve symmetric monoidal structure, sending symmetric monoidal categories to symmetric monoidal **SLat**-categories and vice versa.

In particular, the 2-functor $\mathcal{F}$ turns every category $\mathbb{C}_0 \in \mathbf{Cat}$ freely into a semilattice-enriched category $\mathbb{C} \in \mathbf{SLat-Cat}$ by making every Hom set of $\mathbb{C}_0$ into the free semilattice on that set along F . Being a 2-functor, it preserves dualities (in the 2-categorical sense): it lifts an adjunction in between categories in **Cat** into an **SLat**-adjunction between their free semilattice-enriched counterparts. We take closure, in the sense of Definition 2.6, to be defined generally for **SLat**-categories by an analogous **SLat**-adjunction $- \otimes B \dashv B - \circ -$ for every object B . Thus, applying the 2-functor $\mathcal{F}$ to $\mathbf{CS}(\Sigma)$ we get a closed symmetric monoidal **SLat**-category $\mathbf{CS}(\Sigma)^+$ by this lifting. We can similarly derive **SLat**-enriched versions of $\Lambda_{A,B,C}$ and $\mathbf{ev}_{A,B}$ morphisms using the lifted adjunction as $\mathbf{ev}_{A,B} = \varepsilon_A$ and $\Lambda_{A,B,C}(f) = \eta_A \circ (B - \circ f)$. The following distributivity law can then be derived, forming the basis of encoding equivalences inside abstractions:

$$\begin{aligned} \Lambda_{A,B,C}(f+g) &= \eta_A \circ (B - \circ (f+g)) = \eta_A \circ (B - \circ f + B - \circ g) \\ &= \eta_A \circ (B - \circ f) + \eta_A \circ (B - \circ g) = \Lambda_{A,B,C}(f) + \Lambda_{A,B,C}(g). \end{aligned} \tag{1}$$

Similar to $\mathbf{CS}(\Sigma)$, $\mathbf{CS}(\Sigma)^+$ can be freely generated from a set of closed Σ^+ -terms as follows. The set of closed Σ^+ -terms is constructed just like the set of closed Σ -terms, with $+$ being an additional constructor for terms. Terms of $\mathbf{CS}(\Sigma)^+$ are quotiented by the laws of closed symmetric monoidal **SLat**-categories. The same applies in the presence of equations for $\mathbf{CS}(\Sigma, \mathcal{E})^+$.

¹Semilattice homomorphisms $\{*\} \rightarrow P$ are in bijection with the elements of P as a set, so this is an element of the underlying set of $\mathbb{C}(A, A)$.

²More generally, if we were to define **SLat**-natural transformations, we could view **SLat-Cat** as a symmetric monoidal 2-category [22].

³For example, given a morphism $f \circ (a+b)$, we know it is equal to $f \circ a + f \circ b$ as per the axioms of the category.

3 String Diagram Rewrite Theory

We recall the fundamental definitions and results of string diagram rewrite theory for symmetric monoidal categories, recapitulating the correspondence between string diagram rewriting and an appropriate notion of double pushout (DPO) rewriting of certain hypergraphs, as established by Bonchi et al. [6, 7].

We write $(-)^*$ for the free monoid monad over **Set**, extended element-wise to relations; for $R \subseteq V \times W$ and $v \in V$ we write $R(v) \subseteq W$ in functional notation. We elide coproduct associativity and unit isomorphisms, write ι_j for the j^{th} injection into $X_1 \sqcup \dots \sqcup X_n$, $[f, g]$ for co-pairing, and $A \sqcup_{f, g} B$ for the pushout of $A \xleftarrow{f} C \xrightarrow{g} B$; we call A, B the *feet* and C the *carrier* of the cospan. Given an ordered sequence of objects $[A_1, \dots, A_n]$, we write $w([A_1, \dots, A_n])$ for the object obtained by tensoring them, $A_1 \otimes \dots \otimes A_n$, with $w([]) = I$ the monoidal unit; e.g. $w([A, B]) = A \otimes B$.

3.1 Hypergraphs

Hypergraphs generalise directed graphs by allowing edges to have multiple (ordered) sources and targets. Bonchi et al. [7] show that a certain class of hypergraphs *with interfaces* are sound and complete for categorical presentations of arbitrary symmetric monoidal theories. They develop the use of hypergraphs as combinatorial representations of string diagrams for symmetric monoidal categories, whereby morphisms (nodes) are represented as hyperedges, and objects (wires) as vertices. String diagram rewriting is then formalised as DPO rewriting of said hypergraphs. We briefly present this correspondence, but the interested reader should consult the work of Bonchi et al. [6, 7].

Definition 3.1 (Category of hypergraphs). A hypergraph $\mathcal{G}$ over a monoidal signature $\Sigma = (\Sigma_O, \Sigma_M)$ is a tuple (V, E, s, t, l) , where V is a finite set of vertices, E is a finite set of hyperedges, $s: E \rightarrow V^*$ is a source function, $t: E \rightarrow V^*$ is a target function, and $l = (l_V: V \rightarrow \Sigma_O, l_E: E \rightarrow \Sigma_M)$ is a pair of labelling functions that assigns each hyperedge and vertex a generator and a type from monoidal signature Σ respectively.

The labelling function must respect the typing of the generator:

$$\forall e \in E. l(e) = c: A \rightarrow B \implies l_V^*(s(e)) = A \text{ and } l_V^*(t(e)) = B.$$

We call a hypergraph discrete if its set of hyperedges is empty.

A *hypergraph homomorphism* $\phi: \mathcal{F} \rightarrow \mathcal{G}$ is given by a pair of functions $\phi_V: V_{\mathcal{F}} \rightarrow V_{\mathcal{G}}, \phi_E: E_{\mathcal{F}} \rightarrow E_{\mathcal{G}}$ which respect source, target, and labelling.

Hyp(Σ) denotes the category of hypergraphs and hypergraph homomorphisms over Σ .

String diagrams also have dangling wires exiting at their top/bottom boundaries, corresponding to special *input* and *output* vertices.

Definition 3.2 (Hypergraphs with interfaces). A hypergraph with interfaces is a cospan $n \xrightarrow{\text{in}} \mathcal{G} \xleftarrow{\text{out}} m$ in **Hyp**(Σ) where n, m are discrete hypergraphs denoting input and output interfaces. We write **HypI**(Σ) for the full subcategory of the quotient category of cospans⁴ of **Hyp**(Σ) whose objects are discrete hypergraphs and morphisms are isomorphism classes of cospans of hypergraphs⁵.

Additional constraints on hypergraphs are needed to put them in bijective correspondence with string diagrams in a symmetric monoidal category: *monogamy*, which enforces that in and out are injective and constrain the in-degree and out-degree of vertices in $\mathcal{G}$, and *acyclicity*, which states that there are no loops in the hypergraph. Together, these ensure that a hypergraph with interfaces “looks like” a string diagram, and determines a subcategory **MHypI**(Σ) $\hookrightarrow$ **HypI**(Σ) — the category of *monogamous directed*

⁴This is the 1-categorical version of the bicategory of cospans: for a bicategory, its quotient category is given by the same objects but by taking *isomorphism classes* of morphisms instead.

⁵By abuse of notation, we ignore the distinction between a cospan and its isomorphism class — this is validated by replacing “pushout” (which is only determined up to unique isomorphism) by “chosen pushouts”, which any concrete setting (e.g. an implementation of DPO rewriting) would satisfy.

acyclic (MDA) hypergraphs with interfaces. An example and a non-example of an MDA cospan can be found in Appendix B.

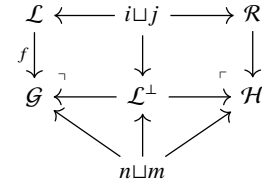
There is a symmetric monoidal equivalence which witnesses that MDA hypergraphs with interfaces are precisely string diagrams.

Theorem 3.3 (Corollary 26 of [7]). $\mathbf{S}(\Sigma) \cong \mathbf{MHypI}(\Sigma)$

The practical meaning of this is that two Σ -terms f and g are equal modulo symmetric monoidal category equations if and only if they are represented as the same hypergraph with interfaces. Appendix D explains the functor underlying this equivalence explicitly. Term rewriting and hence equality modulo $\mathcal{E}$ is then implemented as DPO rewriting over such hypergraphs with interfaces, which we recall.

3.2 DPOI-Rewriting for Hypergraphs with Interfaces

The rewriting of hypergraphs with interfaces (and hence of string diagrams) is formalised as double-pushout rewriting with interfaces (DPOI) [6, 7], whose defining diagram is shown on the right. A rewrite rule is a pair of discrete cospans with matching interfaces, $i \rightarrow \mathcal{L} \leftarrow j$ and $i \rightarrow \mathcal{R} \leftarrow j$, equivalently encoded as a single span $\mathcal{L} \leftarrow i \sqcup j \rightarrow \mathcal{R}$ [6, Remark 3.14]; we make this identification throughout. Applying the



rule to $\mathcal{G}$ (with interface $n \sqcup m$) proceeds in two pushout steps. A *match* $f: \mathcal{L} \rightarrow \mathcal{G}$ locates an occurrence of $\mathcal{L}$; the left pushout square computes the *pushout complement* $\mathcal{L}^\perp$ — what remains of $\mathcal{G}$ after excising the matched part, with both $n \sqcup m$ and $i \sqcup j$ contributing to the interface, the latter corresponding to newly created inputs and outputs of the *hole*. The right pushout square then glues $\mathcal{R}$ into this context, producing the rewritten graph $\mathcal{H}$, with the interface $n \sqcup m$ threaded through both steps.

Definition 3.4 (DPOI rewriting). Given a span of morphisms $\mathcal{G} \leftarrow n \sqcup m \rightarrow \mathcal{H}$ in $\mathbf{Hyp}(\Sigma)$, we say $\mathcal{G}$ rewrites to $\mathcal{H}$ with interface $n \sqcup m$ via rewrite rule $\mathcal{L} \leftarrow i \sqcup j \rightarrow \mathcal{R}$ if there exists an object $\mathcal{L}^\perp$ and morphisms completing the commutative diagram above such that the two marked squares are pushouts.

For soundness and completeness with respect to Σ -term rewriting modulo SMC, matches are restricted to *convex* matches and pushout complements to *boundary complements* (see Appendix B). Bonchi et al. [7] show that any two Σ -terms f and g are equal modulo SMC and $\mathcal{E}$ if and only if $\llbracket f \rrbracket \Rightarrow \llbracket g \rrbracket$, where $\llbracket - \rrbracket$ is the functor from Theorem 3.3 and $\llbracket \mathcal{E} \rrbracket = \{ \langle \llbracket l \rrbracket, \llbracket r \rrbracket \rangle, \langle \llbracket r \rrbracket, \llbracket l \rrbracket \rangle \mid l = r \in \mathcal{E} \}$. Next, we build a combinatorial representation for closed Σ^+ -terms to obtain a notion of DPOI rewriting that also subsumes equality modulo the laws of closed symmetric monoidal **SLat**-categories.

4 Closed E-Hypergraphs

Tiurin et al. [33] introduced *e-hypergraphs* — a combinatorial representation of morphism terms for free symmetric monoidal **SLat**-categories with a modified notion of DPOI rewriting to make it sound and complete for equalities between morphism terms. The **SLat**-enrichment structure was encoded using hierarchical hyperedges in a hypergraph following the approach of a well-established body of literature on hierarchical hypergraphs [14, 10, 27, 9, 20, 5]. The last work in loc. cit. introduced a combinatorial representation of closed Σ -terms (without enrichment). Notably, the laws of a **SLat**-enrichment used by Tiurin et al. [33] and the laws of Definition 2.6 [5] were not absorbed by the combinatorial representation and instead were implemented on top as structural DPOI rewrites.

We follow an analogous approach by combining combinatorial representations of Alvarez-Picallo et al. [5] and Tiurin et al. [33]. We next extend the definition of e-hypergraphs for symmetric monoidal **SLat**-categories [33] to accommodate new structure brought by closure.

Definition 4.1. Over a closed symmetric monoidal signature $\Sigma = (\Sigma_O, \Sigma_M)$, we define a closed e-hypergraph $\mathcal{G}$ as a tuple

$$(V, E, s, t, l_V, l_E, <, <^\mu, \smile), \text{ where}$$

- V is a set of vertices.
- $E = \mathbf{E} \cup \mathbf{E} \cup \mathbf{E}$ is a set of hyperedges that is formed of three disjoint sets of hyperedges defined below.
- $s, t: E \rightarrow V^*$ are source and target functions.
- $l_E: E \rightarrow \Sigma_M \sqcup 1$, $l_V: V \rightarrow \Sigma_O \cup \text{obj}_\perp$ are label functions, where $\text{obj}_\perp$ consists of objects $A \multimap B$ for $A, B \in \text{obj}_{\Sigma_O}$.
- $< \subseteq \mathbf{E} \times (V \sqcup E)$ and $<^\mu \subseteq \mathbf{E} \times (V \sqcup E)$ with $\mathbf{E}, \mathbf{E} \subset E$ are disjoint partial orders that are defined as a transitive closure of corresponding immediate predecessor relations that we denote as $<^\mu$ and $<^\mu$. For $x \in V \sqcup E$, the set of predecessors of x is denoted as $[x] := \{x' \mid \exists y_1 = x', \dots, y_n = x, y_i <^\mu y_{i+1} \text{ or } y_i <^\mu y_{i+1}\}$. We call edges e such that $l(e) = \perp$ hierarchical and edges e and vertices v such that $[e] = \emptyset$ and $[v] = \emptyset$ top-level. Each of these relations should satisfy the following:
 1. each set of predecessors consists exclusively of hierarchical edges;
 2. each x has at most one immediate predecessor;
 3. edges e such that there is no x such that $e <^\mu x$ or $e <^\mu x$ are labelled;
 4. the relations are closed under connectivity, i.e., if $v \in s(e)$ then $e' <^\mu e$ if and only if $e' <^\mu v$, similarly for $v \in t(e)$ and $<^\mu$.
- $\smile$ is a consistency relation which is given by the union of a family of equivalence relations $\smile_p$ on each set $\{x \in V \sqcup E \mid p <^\mu x\}$ of elements which share the same parent where each relation is also closed under connectivity, i.e. if $v \in s(e)$ or $v \in t(e)$ such that $p <^\mu (v)$ and $p <^\mu (e)$ then $v \smile_p e$. We require that $\smile_p \neq (E_p \sqcup V_p) \times (E_p \sqcup V_p)$ where $V_p = \{v \mid p <^\mu v\}$ and $E_p = \{e \mid p <^\mu e\}$.

$\mathbf{E}$ and $\mathbf{E}$ are then induced by the corresponding relations and consist of exclusively hierarchical edges, so $\mathbf{E} = E \setminus \mathbf{E} \cup \mathbf{E}$.

Intuitively, edges from $\mathbf{E}$ encode equivalence classes, i.e. e-classes, while edges from $\mathbf{E}$ encode λ -abstraction boxes. Also note the typing on the l_V function. We work with freely generated closed **SLat**-categories, and to avoid unnecessary complications we limit the labelling of vertices to such a type. This prohibits the labelling of vertices by e.g. $A \otimes B$, but allows them to be labelled by $A \otimes B \multimap C \otimes D$. $<$ defines closed monoidal structure as defined by Alvarez-Picallo et al. [5] and $(<, \smile)$ defines **SLat** structure as per Tiurin et al. [33].

Consider an example in the middle part of Figure 6a. The closed e-hypergraph there is given by $V = \{v_1, \dots, w_{10}\}$, $E = \{e_1, \dots, e_7\}$, $\mathbf{E} = \{e_1\}$, $\mathbf{E} = \{e_2\}$, $l_E = e_1 \mapsto \perp, e_2 \mapsto \perp, e_3 \mapsto g, \dots, e_7 \mapsto f$; the elements of the involved relations are shown in Figure 6b, and, in particular, we have $[e_5] = \{e_2, e_1\}$. $\smile_{e_1}$ defines a partition on immediate successors of e_1 which we delimit by a dashed line. We depict edges in $\mathbf{E}$ with a dashed box, and edges in $\mathbf{E}$ with a round box.

Definition 4.2 (Closed e-hypergraph homomorphism). A homomorphism $\phi: \mathcal{F} \rightarrow \mathcal{G}$ between closed e-hypergraphs $\mathcal{F}, \mathcal{G}$ is a pair of functions $\phi_V: V_{\mathcal{F}} \rightarrow V_{\mathcal{G}}, \phi_E: E_{\mathcal{F}} \rightarrow E_{\mathcal{G}}$ such that:

1. ϕ is a hypergraph homomorphism;
2. $\phi_E(\mathbf{E}_{\mathcal{F}}) \subseteq \mathbf{E}_{\mathcal{G}}$ and $\phi_E(\mathbf{E}_{\mathcal{F}}) \subseteq \mathbf{E}_{\mathcal{G}}$;
3. for $v \in V_{\mathcal{F}}$ and $e \in E_{\mathcal{F}}$

$$\text{if } e <_{\mathcal{F}}^\mu v \text{ then } \phi(e) <_{\mathcal{G}}^\mu \phi(v), \quad \text{if } e <_{\mathcal{F}}^\mu v \text{ then } \phi(e) <_{\mathcal{G}}^\mu \phi(v),$$
 and for $e_1, e_2 \in E_{\mathcal{F}}$

$$\text{if } e_1 <_{\mathcal{F}}^\mu e_2 \text{ then } \phi(e_1) <_{\mathcal{G}}^\mu \phi(e_2), \quad \text{if } e_1 <_{\mathcal{F}}^\mu e_2 \text{ then } \phi(e_1) <_{\mathcal{G}}^\mu \phi(e_2);$$
4. for all $x_1, x_2 \in \mathcal{F}$, $x_1 \smile_{\mathcal{F}} x_2$ implies $\phi(x_1) \smile_{\mathcal{G}} \phi(x_2)$.

The conditions on closed e-hypergraph homomorphisms require preserving immediate predecessors and the consistency relation.

Definition 4.3 (Category of closed e-hypergraphs). The category of closed e-hypergraphs **CEHyp**(Σ) has closed e-hypergraphs as objects and closed e-hypergraph homomorphisms as morphisms.

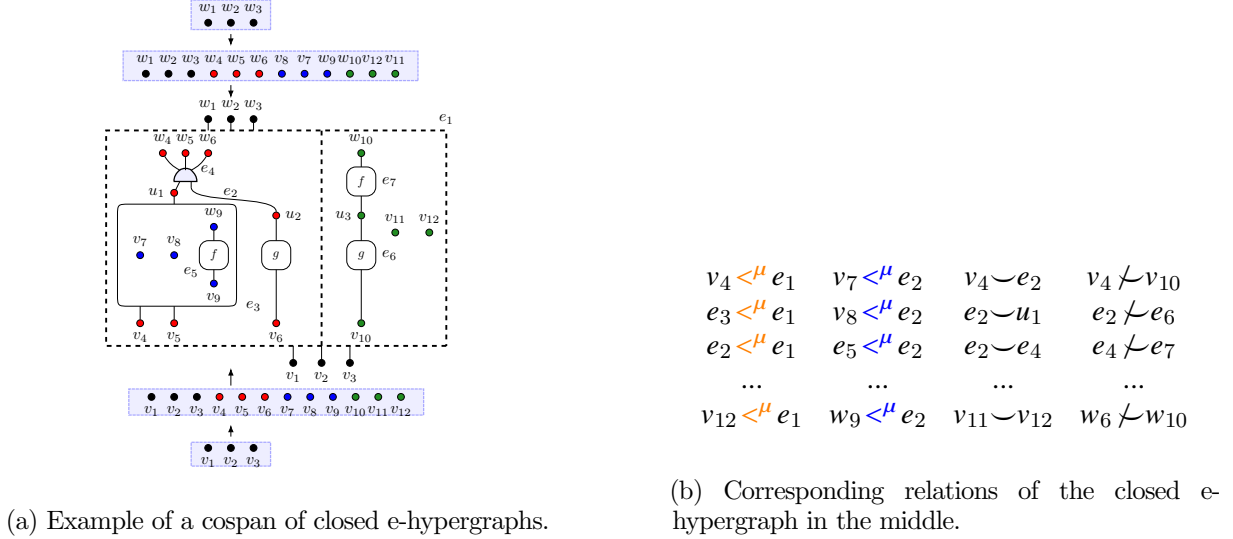


Figure 6

The category of closed e-hypergraphs has coproducts, given by the disjoint union of closed e-hypergraphs, and an initial object given by the empty closed e-hypergraph. A concrete description of the pushout of two morphisms in this category is given in Appendix C. Generally, the pushout of two closed e-hypergraph homomorphisms need not exist, but it does when certain technical conditions are satisfied; in particular, a pushout for the composition of two cospans with discrete feet always exists.

As in the case of $\mathbf{S}(\Sigma)$ and $\mathbf{MHypI}(\Sigma)$, closed e-hypergraphs require interfaces in order to model string diagrams with dedicated inputs and outputs. Because string diagrams for closed symmetric monoidal $\mathbf{SLat}$ -categories are nested, containing substring diagrams inside e-class boxes and λ -abstraction boxes, we need to account for the interfaces in these nested contexts as well. Nested inputs and outputs of a string diagram do not participate in composition of string diagrams, hence they need to be distinguished from the ones which do. We thus propose *extended* cospans of discrete closed e-hypergraphs and use the notation $n \setminus m$ to denote the discrete closed e-hypergraph with $n \setminus m$ vertices, in particular with the vertices of m removed from n when vertices of m is a sub closed e-hypergraph of n .

Definition 4.4 (Category of closed e-hypergraphs with interfaces). The category of closed e-hypergraphs with extended interfaces $\mathbf{CEHypI}(\Sigma)$ has discrete *ordered* closed e-hypergraphs as objects, with Hom sets $\mathbf{CEHypI}(\Sigma)(n, m)$ consisting of isomorphism classes (Definition 4.5) of extended cospans, defined as:

$$n \xrightarrow{f_{ext}} n' \xrightarrow{f_{int}} \mathcal{G} \xleftarrow{g_{int}} m' \xleftarrow{g_{ext}} m,$$

where $\mathcal{G}$ is a closed e-hypergraph, and n, n', m, m' are discrete ordered closed e-hypergraphs, f_{ext}, g_{ext} are monomorphisms in $\mathbf{CEHyp}(\Sigma)$, and the image of $f_{ext} \circ f_{int}$ and of $g_{ext} \circ g_{int}$ consist exclusively of top-level vertices, and the vertices in the strictly internal interface (defined below) are not top-level. Write $\mathcal{G}$ to denote the extended cospan when it is clear from context $\mathcal{G}$ is equipped with extended interfaces.

We call n *external input interfaces*, n' *internal input interfaces*, and $n' \setminus f_{ext}(n)$ the *strictly internal input interfaces*. We do analogously for the *output interfaces*, with respect to m, m' and $m' \setminus g_{ext}(m)$. We conflate f_{ext} with $f_{ext} \circ f_{int}$ when it is clear from context, and also conflate n and m with their images in n' and m' , and likewise n, m, n', m' with their images in $\mathcal{G}$. Given an edge $e \in E_{\mathcal{G}}$ such that $l(e) = \perp$, we call the *inputs of e* the intersection of the strictly internal input interface of $\mathcal{G}$ with the immediate successors of e , and analogously for the *outputs of e* . Graphically, we depict such extended cospans as in Figure 6a: feet of the cospan are depicted within shaded regions. All non-black vertices form strictly internal interfaces. We follow the same orientation that we chose for string diagrams — the

input interface of the cospan is at the bottom and the output interface is at the top.

Definition 4.5 (Isomorphic cospans). Consider a relation

$$R = \{xRy \text{ if } f_{\text{int}}(x) \smile f_{\text{int}}(y) \text{ or } \exists z \text{ s.t. } z <^\mu f_{\text{int}}(x) \text{ and } z <^\mu f_{\text{int}}(y)\}$$

for $x, y \in n'$, with S its reflexive closure. The latter partitions n' into non-empty subsets $\{p_j\}_{j=1}^k$. We get an analogous partition for m' . In Figure 6a, colours denote the partitions for input and output internal interfaces. Two extended cospans are isomorphic if there exist isomorphisms α, β and γ making the following diagram commute and such that α and γ preserve order within each p_j .

$$\begin{array}{ccccc} n & \xrightarrow{\quad} & n' & \xrightarrow{\quad} & \mathcal{G} & \xleftarrow{\quad} & m' & \xleftarrow{\quad} & m \\ & \searrow & \downarrow \alpha & & \downarrow \beta & & \downarrow \gamma & & \searrow \\ & & n'' & \xrightarrow{\quad} & \mathcal{G}' & \xleftarrow{\quad} & m'' & \xleftarrow{\quad} & m \end{array}$$

Intuitively, such notion of isomorphism makes it possible to freely rearrange interfaces for disjoint components. For example, in the input interface of cospan in Figure 6a, we can swap $v_4v_5v_6$ with $v_7v_8v_9$ but we can not swap v_4v_5 with each other as it would result in a non-isomorphic cospan.

Composition of two morphisms $n \xrightarrow{f_{\text{ext}}} n' \xrightarrow{f_{\text{int}}} \mathcal{F} \xleftarrow{f'_{\text{int}}} m' \xleftarrow{f'_{\text{ext}}} m$ and $m \xrightarrow{g_{\text{ext}}} m'' \xrightarrow{g_{\text{int}}} \mathcal{G} \xleftarrow{g'_{\text{int}}} k' \xleftarrow{g'_{\text{ext}}} k$ is computed in two stages. First, $\mathcal{H}$ is computed as the result of the pushout square shown on the right; then, the result of composition is defined as follows:

$$\begin{array}{ccccccc} n & \xrightarrow{f_{\text{ext}}} & n' & \xrightarrow{f_{\text{int}}} & \mathcal{F} & \xleftarrow{f'_{\text{int}}} & m' \xleftarrow{f'_{\text{ext}}} m \\ & \searrow & \downarrow & & \downarrow & & \downarrow \\ & & n'' & \xrightarrow{g_{\text{int}}} & \mathcal{G} & \xleftarrow{g'_{\text{int}}} & k' \xleftarrow{g'_{\text{ext}}} k \\ & & \swarrow p_1 & & \swarrow p_2 & & \\ & & \mathcal{H} & & \end{array}$$

$$n \xrightarrow{f_{\text{ext}} \circ \iota_1} n' \sqcup (m'' \setminus m) \xrightarrow{h_1} \mathcal{H} \xleftarrow{h_2} k' \sqcup (m' \setminus m) \xleftarrow{g'_{\text{ext}} \circ \iota_1} k$$

where h_i are defined as follows, using $|$ to denote the restriction of a function on a discrete closed e-hypergraph.

$$h_1 = [f_{\text{int}} \circ p_1, (g_{\text{int}} \circ p_2)|_{m'' \setminus m}] \quad h_2 = [g'_{\text{int}} \circ p_2, (f'_{\text{int}} \circ p_1)|_{m' \setminus m}]$$

The identity of composition is given by the obvious extended cospan of identities. **CEHypI**(Σ) inherits a symmetric monoidal structure from the coproduct (and initial object) structure of **CEHyp**(Σ), analogously to Definition 3.2.

For the same reasons as in the case of **S**(Σ) we restrict cospans to MDA cospans. We also need to enforce typing constraints on hierarchical edges so that they represent well-typed closed Σ^+ -terms.

Definition 4.6. We call a cospan $n \xrightarrow{f_{\text{ext}}} n' \xrightarrow{f_{\text{int}}} \mathcal{G} \xleftarrow{g'_{\text{int}}} m' \xleftarrow{g_{\text{ext}}} m$ monogamous directed acyclic if

- underlying hypergraph of $\mathcal{G}$ is directed acyclic;
- in- and out- degrees of every vertex is at most 1;
- f_{int} and g_{int} are monomorphisms;
- vertices with in-degree (respectively, out-degree) of 0 are precisely the image of f_{int} (respectively, g_{int}).

MDA cospans and discrete closed e-hypergraphs form a category **MCEHypI**(Σ).

Definition 4.7. We call a monogamous cospan

$$n \xrightarrow{f_{\text{ext}}} n' \xrightarrow{f_{\text{int}}} \mathcal{G} \xleftarrow{g'_{\text{int}}} m' \xleftarrow{g_{\text{ext}}} m$$

well-typed if all hierarchical edges of $\mathcal{G}$ are well-typed:

- for each $e \in E$ consider sets I and O of its input and output vertices partitioned according to $\smile_e$; e is well-typed if for each element S of the partition of I $w(l_V^*(S)) = w(l_V^*(s(e)))$ and similarly for O and $t(e)$;
- for each $e \in E$ consider sets I and O of its input and output vertices; e is well-typed if there exists an object $B \in \text{obj}_{\Sigma_O}$ such that $w(l_V^*(s(e))) \otimes B = w(l_V^*(I))$ and $B \multimap w(l_V^*(O)) = w(l_V^*(t(e)))$.

We denote the category of well typed MDA cospans as **MCEHypI**(Σ).

Again, considering Figure 6a, to make edge e_1 well-typed, it is necessary that $l_V^*([v_4, v_5, v_6]) = l_V^*([v_{10}, v_{11}, v_{12}]) = l_V^*([v_1, v_2, v_3])$ and $l_V^*([w_4, w_5, w_6]) = l_V^*([w_{10}, v_{12}, v_{11}]) = l_V^*([w_1, w_2, w_3])$ (note

the change of ordering in the output of e_1). To make edge e_2 well-typed, it is necessary that $l_V^*([v_4, v_5, v_9]) = l_V^*([v_7, v_8, v_9])$ and $l_V([u_1]) = l_V([v_9]) \multimap l_V^*([v_8, v_7, w_9])$.

5 DPOI-Rewriting for Closed E-Hypergraphs

Because extended cospans have a more general notion of interface, including *internal* vertices, DPOI rewriting as presented in Section 3 needs some adjustments, which we define in this section. The construction in this section closely follows the one introduced by Tiurin et al. [33] with some adjustments to account for $<$. We do not expect internal interfaces to be preserved during rewriting: e.g. when the semilattice equations are considered as rewrites. Thus, we wish a rewrite rule to be a pair of *extended* cospans of closed e-hypergraphs with matching *external* (but not necessarily internal) interfaces:

$$n \rightarrow n' \rightarrow \mathcal{L} \leftarrow m' \leftarrow m, \quad n \rightarrow n'' \rightarrow \mathcal{R} \leftarrow m'' \leftarrow m.$$

Analogously to Section 3, observing that the cospan $0 \rightarrow 0 \rightarrow \mathcal{L} \leftarrow n' \sqcup m' \leftarrow n \sqcup m$ expresses the same data as $n \rightarrow n' \rightarrow \mathcal{L} \leftarrow m' \leftarrow m$ allows us to encode rewrite rules as extended cospans of the form $\mathcal{L} \leftarrow n' \sqcup m' \leftarrow n \sqcup m \rightarrow n'' \sqcup m'' \rightarrow \mathcal{R}$, which fits into the DPO formalism. We implicitly make this identification from hereon.

Before introducing *extended* DPOI rewriting, the definition of boundary complement must guarantee that rewriting yields a monogamous directed acyclic closed e-hypergraph. First, we introduce the ancillary notion of a *down-closed* graph:

Definition 5.1 (Convex down-closed subgraph). We call a sub closed e-hypergraph $\mathcal{G}$ of $\mathcal{H}$ down-closed if for all $e \in E_{\mathcal{G}}$, all children of e are also in $\mathcal{G}$. A sub e-hypergraph $\mathcal{G}$ is convex, if for all $v, w \in V_{\mathcal{G}}$, any directed path from v to w in $\mathcal{H}$ consists entirely of vertices and edges in $\mathcal{G}$.

Definition 5.2 (Extended boundary complement). For MDA cospans $i \rightarrow i' \rightarrow \mathcal{L} \leftarrow j' \leftarrow j$ and $n \rightarrow n' \rightarrow \mathcal{G} \leftarrow k' \leftarrow k$ and mono $m: \mathcal{L} \rightarrow \mathcal{G}$ in $\mathbf{CEHyp}(\Sigma)$, a pushout complement $i \sqcup j \rightarrow \mathcal{L}^\perp \rightarrow \mathcal{G}$ in the square of

$$\begin{array}{ccc} \mathcal{L} & \xleftarrow{[f'_{int}, g'_{int}]_{i'}} \sqcup \xleftarrow{[f'_{ext}, g'_{ext}]_{j'}} & j \\ m \downarrow & & \downarrow [c_1, c_2] \\ \mathcal{G} & \xleftarrow{l} & \mathcal{L}^\perp \\ [g_{int}, h_{int}]_{n'} \sqcup k' \swarrow & & \nwarrow [d_1, d_2] \\ & n \sqcup k & \end{array}$$

is a *boundary complement* if:

1. $m(\mathcal{L})$ is a convex down-closed closed e-hypergraph;
2. $[c_1, c_2]$ is a monomorphism;
3. for all v, w in $\mathcal{G}$ in the image of $i \sqcup j$, v and w share the same sequence of predecessors, and either v, w are top-level, or $v \smile w$, or there exists $e \in E$ s.t. $v, w <^\mu e$;
4. for all v, w in $\mathcal{L}^\perp$ in the image of $i \sqcup j$, v and w share the same sequence of predecessors, and either v, w are top-level or $v \smile w$, or there exists $e \in E$ s.t. $v, w <^\mu e$;
5. there exist $d_1: n \rightarrow \mathcal{L}^\perp$ and $d_2: k \rightarrow \mathcal{L}^\perp$ making the above triangle commute; and
6. if the image of $\mathcal{L}$'s external interfaces under m consists exclusively of top-level vertices of $\mathcal{G}$ then there exists a *well-typed* extended MDA cospan

$$n \sqcup j \xrightarrow{f_1 \sqcup id_j} n' \setminus (i' \setminus i) \sqcup j \xrightarrow{[g_1, c_2]} \mathcal{L}^\perp \xleftarrow{[g_2, c_1]} k' \setminus (j' \setminus j) \sqcup i \xleftarrow{f_2 \sqcup id_i} k \sqcup i$$

7. if the image of $\mathcal{L}$'s external interfaces under m consists exclusively of not top-level vertices of $\mathcal{G}$ then there exists a *not necessarily well-typed* extended MDA cospan

$$n \xrightarrow{f_1} n' \setminus (i' \setminus i) \sqcup j \xrightarrow{[g_1, c_2]} \mathcal{L}^\perp \xleftarrow{[g_2, c_1]} k' \setminus (j' \setminus j) \sqcup i \xleftarrow{f_2} k$$

where f_i and g_i are defined as follows. The strictly internal interface $i' \setminus i$ of $\mathcal{L}$ is mapped to the internal interface n' of $\mathcal{G}$, since $\mathcal{L}$ is a down-closed subgraph of $\mathcal{G}$, inducing an identification of $i' \setminus i$ in n' . Then map f_1 is given by g_{ext} with its codomain restricted to $n' \setminus (i' \setminus i)$.⁶ The map g_1 is derived from the restriction of g_{int} to type $n' \setminus (i' \setminus i) \rightarrow \mathcal{G}$ by further observing that $\mathcal{L}^\perp$ can be identified within $\mathcal{G}$ — and in particular has the internal interface n' of $\mathcal{G}$ minus $i' \setminus i$. The maps f_2 and g_2 are defined similarly.

Intuitively, the above definition means that when the occurrence of $\mathcal{L}$ within $\mathcal{G}$ is top-level then $\mathcal{L}$'s external interfaces become the part of $\mathcal{L}^\perp$'s external interfaces and when the occurrence is nested, those interfaces become the part of $\mathcal{L}^\perp$'s strict internal interfaces. In both cases when removing the occurrence of $\mathcal{L}$ from $\mathcal{G}$, $\mathcal{L}$'s strict internal interfaces are removed from strict internal interfaces of $\mathcal{G}$ to form internal interfaces of the corresponding MDA cospans above.

The conditions (3) and (4) require that top-level connected components in $\mathcal{L}$ do not end up inside edges of different types within $\mathcal{G}$ and $\mathcal{L}^\perp$ under m and $[c_1, c_2]$. Conditions (2) and (7) are the original boundary complement conditions (forgetting the extra interfaces) introduced by Bonchi et al. [7]. Intuitively these conditions ensure that inputs (respectively, outputs) of $\mathcal{R}$ are glued to the outputs (respectively, inputs) of $\mathcal{L}^\perp$. Convexity and down-closed-ness conditions ensures that the subgraph corresponds to a proper subterm modulo SMC.

Proposition 5.3. The boundary complement of Definition 5.2 when it exists is unique.

We are now ready to define *convex extended DPOI (EDPOI) rewriting* for $\mathbf{CEHyp}(\Sigma)$. It is analogous to Definition B.7, except we must construct internal interfaces explicitly. More precisely, when removing the occurrence of $\mathcal{L}$ from $\mathcal{G}$, *i.e.*, by computing the pushout complement, the internal interfaces of the resulting closed e-hypergraph should be modified to *exclude* the vertices corresponding to $\mathcal{L}$'s strictly internal interfaces (since the vertices they map to have been removed). Then, when gluing $\mathcal{R}$ into the hole, the internal interfaces of the resulting closed e-hypergraph should be modified to *include* the strictly internal interfaces of $\mathcal{R}$ (since new internal interfaces for them to map to have been added).

Definition 5.4 (Convex EDPOI rewriting). Given an extended span of morphisms $\mathcal{G} \leftarrow n' \sqcup k' \leftarrow n \sqcup k \rightarrow n'' \sqcup k'' \rightarrow \mathcal{H}$ in $\mathbf{CEHyp}(\Sigma)$, we say $\mathcal{G}$ *rewrites (convexly) to* $\mathcal{H}$ (with external interface $n \sqcup k$ and taking internal interface $n' \sqcup k'$ to $n'' \sqcup k''$) — denoted by $\mathcal{G} \Rightarrow \mathcal{H}$ — via a rewrite rule $\mathcal{L} \leftarrow i' \sqcup j' \leftarrow i \sqcup j \rightarrow i'' \sqcup j'' \rightarrow \mathcal{R}$ if there exists an object $\mathcal{C}$ and morphisms which complete the following commutative diagram

$$\begin{array}{ccccc}
\mathcal{L} & \leftarrow i' \sqcup j' & \leftarrow i \sqcup j & \longrightarrow i'' \sqcup j'' & \rightarrow \mathcal{R} \\
m \downarrow \sqsupset & & \downarrow & & \sqsupset \downarrow \\
\mathcal{G} & \xleftarrow{n' \sqcup k'} & \mathcal{C} & \xrightarrow{n'' \sqcup k'' [f_1, f_2]} & \mathcal{H} \\
& & \uparrow & & \\
& & n \sqcup k & &
\end{array}$$

such that the following conditions hold:

1. $i \sqcup j \rightarrow \mathcal{C} \rightarrow \mathcal{G}$ is a boundary complement;
2. the internal interfaces of $\mathcal{H}$ are such that: $n'' = n' \setminus (i' \setminus i) \sqcup (i'' \setminus i)$ and $k'' = k' \setminus (j' \setminus j) \sqcup (j'' \setminus j)$
3. the map $f_1 = [g_1, h_1]: n'' \rightarrow \mathcal{H}$ in the diagram above consists of g_1 as defined in Definition 5.2 of boundary complement, and h_1 which is the restriction of the composite $i'' \rightarrow \mathcal{R} \rightarrow \mathcal{H}$ to $i'' \setminus i$, and similarly for the map $f_2: k'' \rightarrow \mathcal{H}$.

Given a set $\mathfrak{R}$ of EDPOI rewrite rules and closed e-hypergraphs with extended interfaces $\mathcal{G}$ and $\mathcal{H}$, we write $\mathcal{G} \Rightarrow_{\mathfrak{R}} \mathcal{H}$ if there exists a EDPOI rewrite in $\mathfrak{R}$ such that via it $\mathcal{G}$ rewrites to $\mathcal{H}$, and we write $\mathcal{G} \Rightarrow_{\mathfrak{R}}^* \mathcal{H}$ for its reflexive, transitive and symmetric closure.

An example of EDPOI rewrite can be seen in Appendix E. The enhanced definition of extended DPOI ensures that the combinatorial representation of morphisms in closed symmetric monoidal $\mathbf{SLat}$ -categories is sound and complete for rewriting of closed Σ^+ -terms.

⁶Noting the image of g_{ext} indeed lies within $n' \setminus (i' \setminus i)$

6 Soundness and completeness

In this section we explore soundness and completeness of EDPOI rewriting with respect to closed Σ^+ -terms. First we recall the rewriting of Σ -terms modulo the laws of symmetric monoidal categories.

Definition 6.1 (Closed Σ -term rewrite rule). A Σ -term rewrite rule is a pair $\langle l, r \rangle$ where l and r are Σ terms with matching inputs and outputs (as lists of input and output types). When l and r are closed Σ -terms, we call such a rewrite rule closed.

In practice such rewrite rules are generated by the equations of the corresponding theory.

Definition 6.2. For Σ -terms, f rewrites to g modulo the laws of symmetric monoidal categories via a rewrite rule $\langle l, r \rangle$, writing $f \rightsquigarrow_{\langle l, r \rangle} g$, if they are representable as $f = c_1 \circ (id_k \otimes l) \circ c_2$ and $g = c_1 \circ (id_k \otimes r) \circ c_2$.

This notion generalises to rewriting of closed Σ -terms.

Definition 6.3. For closed- Σ -terms f, g , and a closed rewrite rule $\langle l, r \rangle$, $f \rightsquigarrow_{\langle l, r \rangle} g$ modulo the laws of symmetric monoidal categories if they are representable as either in Definition 6.2, or as $f = c_1 \circ (id_k \otimes d) \circ c_2$ and $g = c_1 \circ (id_k \otimes e) \circ c_2$ and $d = \Lambda(f')$, $e = \Lambda(g')$ and $f' \rightsquigarrow_{\langle l, r \rangle} g'$.

Lemma 6.4. Every closed Σ^+ -term f can be represented as $f_1 + \dots + f_n$ modulo the laws of symmetric monoidal **SLat**-categories and distributivity law (Equation 1) where none of f_i contain the join operator.

This leads to the following, for rewriting for closed Σ^+ -terms.

Definition 6.5. For closed Σ^+ -term f, g , and a closed Σ -term rewrite rule $\langle l, r \rangle$ $f \rightsquigarrow_{\langle l, r \rangle} g$ modulo the laws of symmetric monoidal **SLat**-categories and distributivity law (Equation 1) if they are representable as $f = f_1 + \dots + f_i + \dots + f_n$ and $g = f_1 + \dots + f_j + \dots + f_n$ such that there is a permutation σ such that there exist indices i, j and $f_{\sigma(i)} \rightsquigarrow f_j$.

Next, to formulate the correspondence between rewriting systems we first define interpretation $\llbracket - \rrbracket$ of closed Σ^+ terms in **MCEHypI**(Σ). It follows by first defining the interpretation of generators by extending the interpretation of plain Σ -terms as morphisms in **MHypI**(Σ) with the interpretation of $\llbracket \text{ev}_{A,B} \rrbracket$ and $\llbracket \Lambda_{A,B,C} \rrbracket$ by following the work of Alvarez-Picallo et al. [5] (Appendix D).

Then we can make **MCEHypI**(Σ) into **SLat**-category by defining a join of two cospans and introducing EDPOI rewrite rules to satisfy the laws of symmetric monoidal **SLat**-categories — one EDPOI rewrite rule for each axiom instance — as was done by Tiurin et al. [33]. Under this interpretation, the cospan from Figure 6a corresponds to the following closed Σ^+ -term $(g \circ f) + ((\Lambda((id \otimes id \circ \text{sym}) \otimes f)) \otimes g) \circ \text{ev}$. Finally, we add a structural rewrite schema rule corresponding to the distributivity law (Equation 1).

We collect all such rewrite schemas into a set $\mathcal{S}$ and then determine a symmetric monoidal **SLat**-category **MCEHypI**(Σ)/ $\mathcal{S}$ by this quotienting. Note that it is not a *closed* symmetric monoidal **SLat**-category. To make it so, we would need to also quotient by the DPO rewrites that arise from the equations in Definition 2.6. We do not perform such quotienting as these equations bear operational meaning, i.e. they model β -reduction when interpreting λ -calculus.

Definition 6.6 (Quotient by rewrites). Given a set of EDPOI rewrite rules $\mathcal{E}$, we denote by **MCEHypI**(Σ)/ $\mathcal{E}$ the **SLat**-category **MCEHypI**(Σ) quotiented by the relation $f \sim g$ if $f \Rightarrow_{\mathcal{E}}^* g$.

This explicit quotienting allows us to only consider cospans of the form $f_1 + \dots + f_n$, similar to Lemma 6.4, such that carrier of f_i does not contain any hierarchical edges from E .

Proposition 6.7 (Proposition 26 and Proposition 27 [5]). For two closed Σ -terms f and g and a closed rewrite rule $\langle l, r \rangle$, $f \rightsquigarrow_{\langle l, r \rangle} g$ if and only if $\llbracket f \rrbracket \Rightarrow_{\langle \llbracket l \rrbracket, \llbracket r \rrbracket \rangle} \llbracket g \rrbracket$, where the latter is defined in a sub-category of **MCEHypI**(Σ) where carriers only contain edges from $E \cup E$.

Proposition 6.8. For two closed Σ^+ -terms f and g and a closed rewrite rule $\langle l, r \rangle$, $f \rightsquigarrow_{\langle l, r \rangle} g$ if and only if $\llbracket f \rrbracket \Rightarrow_{\langle \llbracket l \rrbracket, \llbracket r \rrbracket \rangle} \llbracket g \rrbracket$ in **MCEHypI**(Σ)/ $\mathcal{S}$.

Proof. By definition, we have $f = f_1 + \dots + f_n$ and $g = f_1 + \dots + f_n$, such that $f_i \rightsquigarrow f_j$ for some i, j . This yields two cospans of the form $\llbracket f_1 \rrbracket + \dots + \llbracket f_n \rrbracket$ for $\llbracket f \rrbracket$ and $\llbracket g \rrbracket$; because every f_i is a Σ -term, and carriers in every $\llbracket f_i \rrbracket$ have no edges from E , we can apply Proposition 6.7. $\square$

Practically, this quotienting means that we perform DPO rewrites modulo the equations of **SLat**-SMC and modulo the distributivity law (Equation 1). We can either rewrite the normal forms as mentioned above that contain all possible redexes, or we can genuinely search for redexes modulo these laws by traversing the e-hypergraph through both types of boxes when searching for a redex, which makes this representation practically viable even though it does not absorb all the possible equations.

The proposition above allows us to reason about theories of closed Σ^+ -terms $\mathbf{CS}(\Sigma, \mathcal{E})^+$. Given a set of equations $\mathcal{E}$ of closed Σ -terms, we construct $\llbracket \mathcal{E} \rrbracket = \{ \langle \llbracket l \rrbracket, \llbracket r \rrbracket \rangle, \langle \llbracket r \rrbracket, \llbracket l \rrbracket \rangle \mid l = r \in \mathcal{E} \}$ and then we can perform DPO rewriting in $\mathbf{MCEHypI}(\Sigma)/S$ which formalises, *e.g.*, the example in Figure 4b.

7 Application — $\lambda_{\text{sub}}^{\approx}$ -calculus

In this section, we show the utility of our theory by demonstrating how it can accommodate the *linear substitution calculus* λ_{sub} of Accattoli et al. [4, § 2]. We summarise the linear substitution calculus as the λ -calculus with an additional term former representing explicit substitution; i.e. terms are generated by the grammar

$$t := x \mid tt \mid \lambda x.t \mid t[x/u],$$

where x is a variable drawn from some countably infinite set, tu is an application, and $\lambda x.t$ is an abstraction, as in standard λ -calculus. The term $t[x/u]$ is an explicit substitution of the variable x for the term u in the term t .

The notions of context $C[-]$, bound and free variable are as usual, with the addition that $\text{fv}(t[x/u]) := \text{fv}(t) \setminus \{x\} \cup \text{fv}(u)$. Write $C\llbracket t \rrbracket$ for the term given by substitution of the hole in the context $C[-]$ for the term t , under the additional restriction that the free variables of t are not captured (via abstraction or explicit substitution) by C . In addition to the β - and η -reduction rules of standard λ -calculus, there are reduction rules to account for this explicit substitution which “perform” the substitution or commute it with β -reduction:

$$\begin{aligned} (\lambda x.t)Lu &\rightarrow_{d\beta} t[x/u]L, \\ C\llbracket x \rrbracket[x/u] &\rightarrow_{ls} C\llbracket u \rrbracket[x/u], \\ t[x/u] &\rightarrow_{gc} t, & x \notin \text{fv}(t), \end{aligned}$$

where L is a finite list of substitutions of the form $[x/u][y/v]\dots$.

In programming terms, this explicit substitution can be used to model **let**-bindings and sharing. For instance, the ML-style term **let** $x = 1$ **in** $\text{plus}(x, x)$ is formally represented as the term $\text{plus}(x, x)[x/1]$ in this calculus⁷. In this case, the variable x is valued in the constant 1, but one could imagine instead that x is to be substituted for the result of some expensive computation which we do not desire to repeat twice unnecessarily — in this sense, the sharing of both operands of the operation plus is captured by the explicit substitution.

Graphical equivalence $\sim$ of λ_{sub} -terms is augmentation of α -equivalence of terms with the additional stipulations that

$$\begin{aligned} t[x/u][y/v] &\sim t[y/v][x/u], & x \notin \text{fv}(v) \wedge y \notin \text{fv}(u), \\ (\lambda y.t)[x/u] &\sim \lambda y.(t[x/u]), & y \notin \text{fv}(u), \\ (tv)[x/u] &\sim t[x/u]v, & x \notin \text{fv}(v), \end{aligned} \tag{*}$$

closed under contexts, transitivity, symmetry, and reflexivity. Put simply, it is the stipulation that **let**-bindings may be freely reordered without changing the meaning of the term⁸. The $\lambda_{\text{sub}}^{\approx}$ -calculus is

⁷Technically speaking, in this case we mean the extension of this calculus with flat natural numbers equipped with a binary operation plus .

⁸Recall that our setting is *pure*, in the sense that no sequencing of effects is meaningful.

given by these graphical equivalence classes of terms with the reduction relation defined by that of the λ_{sub} -calculus modulo graphical equivalence.

This calculus is explicitly designed to capture the representation of terms as proof nets: for two terms t, u in the λ_{sub} -calculus, $t \sim u$ if and only if both map to the same proof net [3].

Proposition 7.1. [18, § 5.1.1] Let *string diagram equivalence* $\approx$ of λ_{sub} -terms be the extension of graph equivalence $\sim$ by taking the symmetric closure by $\rightarrow_{ls}$ and $\rightarrow_{gc}$, thus identifying terms under elimination of explicit substitution. It defines the corresponding $\lambda_{\text{sub}}^\approx$ -calculus. We then have that $t \approx u$ if and only if both map to the same string diagram.

This allows us to exploit the string diagram setting of e-hypergraphs to absorb many⁹ of the reductions arising due to the equivalence $\approx$ into our representation rather than explicitly providing them as equations attached to a closed symmetric monoidal theory presenting the $\lambda_{\text{sub}}^\approx$ -calculus.

We will define a monoidal signature whose closed symmetric monoidal theory is the $\lambda_{\text{sub}}^\approx$ -calculus extended by tupling of terms and flat natural numbers equipped with basic arithmetic operations. Let $\Sigma := (\{\mathbf{N}\}, \{\Psi, \P\} \cup \mathbf{N} \cup \{\text{plus}, \text{mult}\}, t)$ where

$$\begin{aligned} t: \{\Psi, \P\} \cup \mathbf{N} \cup \{\text{plus}, \text{mult}\} &\rightarrow \{\mathbf{N}\}^* \times \{\mathbf{N}\}^* \\ t(n) &= ((), (\mathbf{N})), & n \in \mathbf{N}, \\ t(o) &= ((\mathbf{N}, \mathbf{N}), (\mathbf{N})), & o \in \{\Psi, \text{plus}, \text{mult}\}, \\ t(\P) &= ((\mathbf{N}), ()). \end{aligned}$$

The operations that the signature Σ avails are copy and delete, denoted by Ψ and $\P$ respectively, constants for each natural number, and the binary arithmetic operations $+$ and $\times$. We interpret the Σ -terms $\llbracket \tau \rrbracket$ as follows: operations as themselves, id_I and id_N as the identity function $\lambda x.x$ on the empty tuple $()$ and $\mathbf{N}$ respectively, $\text{sym}_{\mathbf{N}, \mathbf{N}}$ as the swap function $\lambda(x, y).(y, x)$, composition $\tau_1 \circ \tau_2$ as the composition function $\lambda x. \llbracket \tau_2 \rrbracket (\llbracket \tau_1 \rrbracket x)$, and $\tau_1 \otimes \tau_2$ as tupling $(\llbracket \tau_1 \rrbracket, \llbracket \tau_2 \rrbracket)$.

Note that, in contrast to conventional presentations of the λ -calculus as a language of (slotted [29, Example 1]) e-graphs [34, Figure 10], here there is no encoding of the term formers for application, abstraction, and explicit substitution. Instead, these constructs are handled ambiently by our semantic setting: any closed symmetric monoidal category (and in particular, $\mathbf{CS}(\Sigma, \mathcal{E})$ for any set of equations $\mathcal{E}$) admits an internal language in the multiplicative fragment of intuitionistic linear logic (a particular type of linear λ -calculus) [2, § 1.7.3]. In other words, with respect to Figure 5, this allows us to *natively* represent abstraction by the rounded boxes of $(\Lambda(f))$, and application by the ev operation with β - and η -conversion arising via equality of morphisms in the category (that is, without having to provide them as data in the form of equations in $\mathcal{E}$). For explicit substitution, we note that the linearity is not desirable; this is why the copy-delete comonoid $(\Psi, \P)$ is present: by Fox’s theorem [17], a Cartesian category is precisely a monoidal one admitting such a *natural* copy-delete structure. Thus, we must endow $\mathcal{E}$ with equations stating that $(\Psi, \P)$ forms a comonoid, and moreover that it is natural: Ψ behaves uniformly as a copy map, and $\P$ behaves as deletion.

Figure 7a shows some programs which are semantically equivalent but syntactically distinct. As alluded to in Subsection 1.4, the traditional (slotted) e-graph approach to modelling the equivalence of these terms (Figure 7b) relies on explicitly encoding substitution operations as e-graph nodes, and their behaviour (the string diagram equivalence relation $\approx$) as rewrite rules, as in Figure 4a — not only is this more complicated, but it also leads to a blow-up in the state space required to perform equality saturation in any extension of the $\lambda_{\text{sub}}^\approx$ -calculus, because we are really rewriting λ_{sub} -terms and

⁹The reason this is not *all* reductions is because e-hypergraphs quotient away only the *structural* isomorphisms of symmetric monoidal closed **SLat**-categories (e.g. associators, unitors, etc.) which generate the “isotopy of string diagrams”; even in the graphical equivalence relation $\sim$, while Equation $*$ is validated in every symmetric monoidal closed **SLat**-category, it is not so by isotopy, but rather the adjunction $-\otimes A \dashv A \multimap -$. In other words, terms floating freely through the boundaries of rounded boxes is validated by our semantics, but produces concretely different e-hypergraphs — this is the purpose of rectification by quotienting along $\mathcal{S}$ in Proposition 6.8, and manifests as additional DPO rewrites to be imposed.

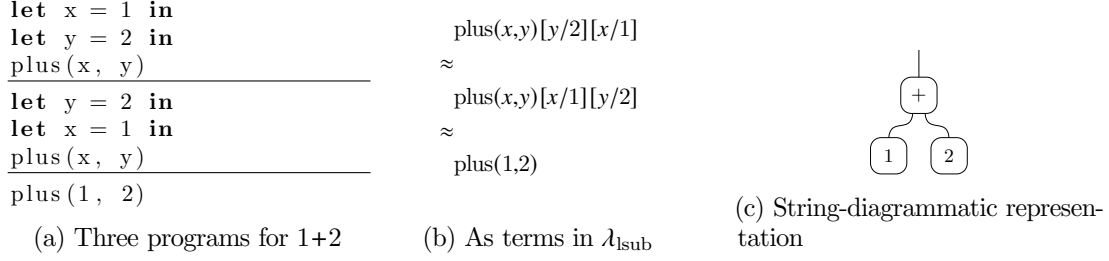
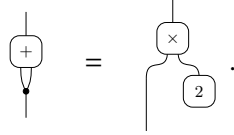


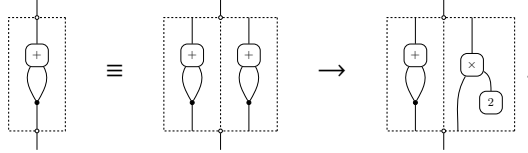
Figure 7: $\approx$ -equivalent terms

computational resources are spent rewriting between equivalent terms (an empirical analysis is provided in Appendix F). In contrast, in our framework, $\lambda_{\text{sub}}^{\approx}$ -terms are directly modelled as string diagrams (Figure 7c¹⁰). We do however note that the slotted approach is more flexible, in the sense that it may model more theories with binders (e.g. π -calculus), whereas our approach is essentially specialised to λ -calculus, relying heavily on the metatheoretic internal language of the ambient setting.

As a second example, we may wish to incorporate the optimisation $\text{plus}(x,x) \rightsquigarrow \text{mult}(x,2)$ in our toy theory. This corresponds to endowing $\mathcal{E}$ with the equation of string diagrams:



Now, in our semantic model, this rewrite can be non-destructively applied to model one step of e-graph saturation:



Here, $\equiv$ is a definitional equality arising from the idempotence in the Hom semilattice, and $\rightarrow$ is an application of DPO rewriting using the previous equation along the right branch. In isolation, this isn't particularly interesting, but the important observation is that this rewriting all happens *locally* in the context of a bigger diagram.

From a practical perspective, the encoding of string diagrams as (e-)hypergraphs provides a normal form for string diagram equivalence classes of terms. As a data structure, this provides an efficient representation upon which to build an implementation of this theory as a fully fledged “e-graphs for string diagram rewriting”, using DPO rewriting as a primitive to implement the standard e-graph operations. We leave this as future work, noting its applications to not only $\lambda_{\text{sub}}^{\approx}$ -calculus, but also other monoidal theories where there are two levels of rewriting:

1. “bureaucratic” reassociation and reparenthesisation, absorbed into isotopy of string diagrams (here, a fragment of string diagram equivalence of λ_{sub} -terms);
 2. algebraic identities arising from the semantics of the theory (e.g. $\text{plus}(x,x) \rightsquigarrow \text{mult}(x,2)$ for arithmetic).
- One such theory is the ZX calculus [11].

References

- [1] M. Abadi, L. Cardelli, P.-L. Curien & J.-J. Lévy (1991): *Explicit substitutions*. *Journal of Functional Programming* 1(4), pp. 375–416, doi:10.1017/S0956796800000186.

¹⁰Note that dashed boxes are elided for convenience here because no rewriting has occurred.

- [2] S. Abramsky & N. Tzevelekos (2010): *Introduction to Categories and Categorical Logic*. In Bob Coecke, editor: *New Structures for Physics*, 813, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 3–94, doi:10.1007/978-3-642-12821-9_1.
- [3] Beniamino Accattoli (2011): *Jumping around the Box: Graphical and Operational Studies on λ -Calculus and Linear Logic*. Ph.D. thesis, La Sapienza University of Rome, Rome, Italy.
- [4] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner & Carlos Lombardi (2014): *A nonstandard standardization theorem*. *ACM SIGPLAN Notices* 49(1), pp. 659–670.
- [5] Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger & Fabio Zanasi (2022): *Rewriting for Monoidal Closed Categories*. In Amy P. Felty, editor: *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel, LIPIcs* 228, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 29:1–29:20, doi:10.4230/LIPIcs.FSCD.2022.29. Available at <https://doi.org/10.4230/LIPIcs.FSCD.2022.29>.
- [6] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski & Fabio Zanasi (2022): *String Diagram Rewrite Theory I: Rewriting with Frobenius Structure*. *J. ACM* 69(2), doi:10.1145/3502719. Available at <https://doi.org/10.1145/3502719>.
- [7] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski & Fabio Zanasi (2022): *String diagram rewrite theory II: Rewriting with symmetric monoidal structure*. *Mathematical Structures in Computer Science* 32(4), pp. 511–541, doi:10.1017/S0960129522000317.
- [8] Francis Borceux (1994): *Handbook of Categorical Algebra II*, pp. 291–348. Encyclopedia of Mathematics and its Applications, Cambridge University Press.
- [9] Roberto Bruni, Fabio Gadducci & Alberto Lluch-Lafuente (2010): *An Algebra of Hierarchical Graphs and its Application to Structural Encoding*. *Sci. Ann. Comput. Sci.* 20, pp. 53–96. Available at <http://www.info.uaic.ro/bin/Annals/Article?v=XX&a=2>.
- [10] Matteo Coccia, Fabio Gadducci & Ugo Montanari (2002): *GS.Lambda Theories: A Syntax for Higher-Order Graphs*. In Richard Blute & Peter Selinger, editors: *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002, Electronic Notes in Theoretical Computer Science* 69, Elsevier, pp. 83–100, doi:10.1016/S1571-0661(04)80560-7. Available at [https://doi.org/10.1016/S1571-0661\(04\)80560-7](https://doi.org/10.1016/S1571-0661(04)80560-7).
- [11] Bob Coecke & Ross Duncan (2011): *Interacting Quantum Observables: Categorical Algebra and Diagrammatics*. *New Journal of Physics* 13(4), p. 043016, doi:10.1088/1367-2630/13/4/043016.
- [12] Geoff SH Cruttwell (2008): *Normed spaces and the change of base for enriched categories*. Ph.D. thesis, Dalhousie University.
- [13] Nicolaas Govert De Bruijn (1972): *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem*. In: *Indagationes mathematicae (proceedings)*, 75, Elsevier, pp. 381–392.
- [14] Frank Drewes, Berthold Hoffmann & Detlef Plump (2002): *Hierarchical Graph Transformation*. *J. Comput. Syst. Sci.* 64(2), pp. 249–283, doi:10.1006/JCSS.2001.1790. Available at <https://doi.org/10.1006/jcss.2001.1790>.
- [15] Oliver Flatt, Samuel Coward, Max Willsey, Zachary Tatlock & Pavel Panchekha (2022): *Small Proofs from Congruence Closure*. pp. 75–83, doi:10.34727/2022/isbn.978-3-85448-053-2_13. Available at <https://repositum.tuwien.at/handle/20.500.12708/81325>. Accepted: 2022-10-12T10:02:40Z.

- [16] Brendan Fong & David I Spivak (2019): *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press.
- [17] Thomas Fox (1976): *Coalgebras and Cartesian Categories*. *Communications in Algebra* 4(7), pp. 665–667, doi:10.1080/00927877608822127.
- [18] Dan Ghica & Fabio Zanasi (2024): *String Diagrams for λ -calculi and Functional Computation*. arXiv:2305.18945. To appear, Cambridge University Press.
- [19] Dan R. Ghica & George Kaye (2023): *Rewriting Modulo Traced Comonoid Structure*. In Marco Gaboardi & Femke van Raamsdonk, editors: *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 260, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 14:1–14:21, doi:10.4230/LIPIcs.FSCD.2023.14. Available at <https://drops.dagstuhl.de/opus/volltexte/2023/17998>. ISSN: 1868-8969.
- [20] Dan R. Ghica, Koko Muroya & Todd Waugh Ambridge (2025): *A robust graph-based approach to observational equivalence*. *Logical Methods in Computer Science* Volume 21, Issue 2:8, doi:10.46298/lmcs-21(2:8)2025. Available at <https://lmcs.episciences.org/8524>.
- [21] William F Keigher (1978): *Symmetric monoidal closed categories generated by commutative adjoint monads*.
- [22] G. M. Kelly, Toby Bartels, William Boshuck, Maria Manuel Clementino, Robert Dawson, Stephen Lack, Tom Leinster, Francisco Marmolejo, Shane O ’conchuir, Craig A. Pastro, Mark Weber, Ralph L. Wojtowicz, G. Max & Kelly Sydney (2022): *BASIC CONCEPTS OF ENRICHED CATEGORY THEORY*. *Elements of ∞ -Category Theory*. Available at <https://api.semanticscholar.org/CorpusID:16424720>.
- [23] Thomas Koehler, Phil Trinder & Michel Steuwer (2022): *Sketch-Guided Equality Saturation: Scaling Equality Saturation to Complex Optimizations of Functional Programs*. arXiv:2111.13040.
- [24] Paul-André Mellès (2006): *Functorial Boxes in String Diagrams*. In Zoltán Ésik, editor: *Computer Science Logic*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–30.
- [25] Aleksandar Milosavljević, Robin Piedeleu & Fabio Zanasi (2023): *String Diagram Rewriting Modulo Commutative (Co)Monoid Structure*. In Paolo Baldan & Valeria de Paiva, editors: *10th Conference on Algebra and Coalgebra in Computer Science (CALCO 2023)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 270, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 9:1–9:17, doi:10.4230/LIPIcs.CALCO.2023.9. Available at <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CALCO.2023.9>.
- [26] Charles Gregory Nelson (1980): *Techniques for program verification*. Stanford University.
- [27] Wojciech Palacz (2004): *Algebraic hierarchical graph transformation*. *J. Comput. Syst. Sci.* 68(3), pp. 497–520, doi:10.1016/S0022-0000(03)00064-3. Available at [https://doi.org/10.1016/S0022-0000\(03\)00064-3](https://doi.org/10.1016/S0022-0000(03)00064-3).
- [28] Amr Sabry & Matthias Felleisen (1992): *Reasoning about programs in continuation-passing style*. In: *Proceedings of the 1992 ACM Conference on LISP and Functional Programming, LFP ’92*, Association for Computing Machinery, New York, NY, USA, pp. 288–298, doi:10.1145/141471.141563. Available at <https://doi.org/10.1145/141471.141563>.
- [29] Rudi Schneider, Marcus Rossel, Amir Shaikhha, Andrés Goens, Thomas Koehler & Michel Steuwer (2025): *Slotted E-Graphs: First-Class Support for (Bound) Variables in E-Graphs*. *Proc. ACM Program. Lang.* 9(PLDI), doi:10.1145/3729326. Available at <https://doi.org/10.1145/3729326>.

- [30] P. Selinger (2010): *A Survey of Graphical Languages for Monoidal Categories*, pp. 289–355. Springer Berlin Heidelberg, doi:10.1007/978-3-642-12821-9_4. Available at http://dx.doi.org/10.1007/978-3-642-12821-9_4.
- [31] Dan Suciu, Yisu Remy Wang & Yihong Zhang (2025): *Semantic foundations of equality saturation*. arXiv:2501.02413.
- [32] Ross Tate, Michael Stepp, Zachary Tatlock & Sorin Lerner (2009): *Equality Saturation: A New Approach to Optimization*. *SIGPLAN Not.* 44(1), pp. 264–276, doi:10.1145/1594834.1480915. Available at <https://doi.org/10.1145/1594834.1480915>.
- [33] Aleksei Tiurin, Chris Barrett, Dan R. Ghica & Nick Hu (2025): *Equivalence Hypergraphs: DPO Rewriting for Monoidal E-Graphs*. In: *2025 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 209–222, doi:10.1109/LICS65433.2025.00023.
- [34] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock & Pavel Panchekha (2021): *Egg: Fast and Extensible Equality Saturation*. *Proc. ACM Program. Lang.* 5(POPL), doi:10.1145/3434304. Available at <https://doi.org/10.1145/3434304>.
- [35] Yihong Zhang, Yisu Remy Wang, Max Willsey & Zachary Tatlock (2022): *Relational e-matching*. *Proceedings of the ACM on Programming Languages* 6(POPL), pp. 1–22, doi:10.1145/3498696. Available at <https://dl.acm.org/doi/10.1145/3498696>.

A SLat

In this section we will define the category of semilattices that we use as a base for enrichment throughout the paper.

Definition A.1 (Semilattice). A *semilattice* $(S, +)$ is a set S equipped with a binary operation $+: S \times S \rightarrow S$ satisfying the following properties:

$$\begin{aligned} a + (b + c) &\equiv (a + b) + c && \text{(associativity)} \\ a + b &\equiv b + a && \text{(commutativity)} \\ a + a &\equiv a && \text{(idempotence)} \end{aligned}$$

Note that we do not require the existence of a unit for $+$, adopting this requirement would mean that every hom-object must be non-empty. Semilattices that satisfy this extra requirement are sometimes called *bounded*, i.e., they are idempotent commutative monoids.

Definition A.2 (Semilattice homomorphism). A homomorphism between two semilattices S_1 and S_2 is a map $\phi: S_1 \rightarrow S_2$ such that $\phi(s +_{S_1} s') = \phi(s) +_{S_2} \phi(s')$ for all $s, s' \in S_1$.

Definition A.3 (Category of semilattices). Semilattices with their respective homomorphisms form a category that we denote **SLat**.

Definition A.4. We define a tensor product of semilattices S_1 and S_2 as follows. $S_1 \otimes S_2$ is the free semilattice on the set $S_1 \times S_2$ — finite non-empty formal joins of pairs (s_1, s_2) with $s_1 \in S_1, s_2 \in S_2$ — quotiented by the following bilinearity relations

- $(s_1 +_{S_1} s'_1, s_2) \equiv (s_1, s_2) +_{S_1 \otimes S_2} (s'_1, s_2)$
- $(s_1, s_2 +_{S_2} s'_2) \equiv (s_1, s_2) +_{S_1 \otimes S_2} (s_1, s'_2)$

Writing $\eta_{S_1, S_2}: S_1 \times S_2 \rightarrow S_1 \otimes S_2$, $(s_1, s_2) \mapsto [(s_1, s_2)]$, for the inclusion of pairs, the tensor product is characterised by the universal property that every *bimorphism* out of $S_1 \times S_2$ —a function $b: S_1 \times S_2 \rightarrow T$ that is a homomorphism in each variable separately—factors as $b = \bar{b} \circ \eta_{S_1, S_2}$ for a unique homomorphism $\bar{b}: S_1 \otimes S_2 \rightarrow T$. On morphisms, given $f: S_1 \rightarrow T_1$ and $g: S_2 \rightarrow T_2$, the composite $\eta_{T_1, T_2} \circ (f \times g)$ is a bimorphism, and we define $f \otimes g: S_1 \otimes S_2 \rightarrow T_1 \otimes T_2$ to be the unique homomorphism with

$$(f \otimes g) \circ \eta_{S_1, S_2} = \eta_{T_1, T_2} \circ (f \times g).$$

The unit for this tensor product is $I = \{*\}$ — a one-element semilattice.

Remark A.5. The action of $\otimes$ on morphisms is functorial: $\text{id} \otimes \text{id} = \text{id}$ and

$$(f \otimes g) \circ (h \otimes k) = (f \circ h) \otimes (g \circ k).$$

Both equations follow from the uniqueness in the universal property, since each side restricts along η to the same bimorphism; for the second,

$$\begin{aligned} (f \otimes g) \circ (h \otimes k) \circ \eta &= (f \otimes g) \circ \eta \circ (h \times k) \\ &= \eta \circ (f \times g) \circ (h \times k) \\ &= \eta \circ ((f \circ h) \times (g \circ k)), \end{aligned}$$

which is exactly the bimorphism defining $(f \circ h) \otimes (g \circ k)$. By the same reasoning the unitors $S \otimes I \cong S$ (given by $(s, *) \mapsto s$), the associators and the symmetry are the unique homomorphisms induced by the evident bimorphisms, and the coherence axioms hold because the corresponding diagrams already commute at the level of bimorphisms, where they reduce to coherence for the cartesian product in **Set**. This equips **SLat** with a symmetric monoidal structure.

Remark A.6. The above definition of a tensor product should not be confused with a tensor given by $\times$, which is constructed in the same vein as the tensor product above with the additional quotienting

$$(s_1, s_2) +_{S_1 \times S_2} (s'_1, s'_2) \equiv (s_1 +_{S_1} s'_1, s_2 +_{S_2} s'_2) .$$

The reason for using the former construction is that it has better theoretical properties, in particular, the free semilattice functor is strong monoidal (corollary A.12).

Definition A.7. For semilattices S_1 and S_2 , the *internal hom* $[S_1, S_2]$ is the set $\mathbf{SLat}(S_1, S_2)$ of homomorphisms $S_1 \rightarrow S_2$, made into a semilattice by the pointwise join

$$(f + g)(x) \triangleq f(x) + g(x).$$

This is well-defined, as $f + g$ is again a homomorphism:

$$\begin{aligned} (f + g)(x + y) &= f(x + y) + g(x + y) \\ &= f(x) + f(y) + g(x) + g(y) \\ &= f(x) + g(x) + f(y) + g(y) \\ &= (f + g)(x) + (f + g)(y). \end{aligned}$$

Proposition A.8. **SLat** is a closed symmetric monoidal category.

Proof. The symmetric monoidal structure is given by the tensor product of definition A.4, with functoriality and coherence established in remark A.5. For closedness, take the internal hom $[S_2, T]$ of definition A.7; the universal property of the tensor gives bijections, natural in S_1 and T ,

$$\mathbf{SLat}(S_1 \otimes S_2, T) \cong \{\text{bimorphisms } S_1 \times S_2 \rightarrow T\} \cong \mathbf{SLat}(S_1, [S_2, T]),$$

where the first is the universal property of $\otimes$ and the second is currying: a bimorphism $b : S_1 \times S_2 \rightarrow T$ corresponds to the homomorphism $s_1 \mapsto b(s_1, -)$ into $[S_2, T]$. Hence $- \otimes S_2 \dashv [S_2, -]$, so **SLat** is closed. $\square$

Remark A.9. The above is a special case of the result by Keigher [21] that says that a category of Eilenberg-Moore algebras induced by a commutative monad on **Set** is canonically symmetric monoidal closed.

Definition A.10. The forgetful functor $U : \mathbf{SLat} \rightarrow \mathbf{Set}$ is given by $\mathbf{SLat}(I_{\mathbf{SLat}}, -) : \mathbf{SLat} \rightarrow \mathbf{Set}$.

Intuitively, the above functor returns the underlying set of a given semilattice S as each morphism from $\{*\} \rightarrow S$ picks out an element of S .

Proposition A.11 (Special case of Proposition 6.4.6 [8]). The forgetful functor $U : \mathbf{SLat} \rightarrow \mathbf{Set}$ has a left adjoint free functor $F : \mathbf{Set} \rightarrow \mathbf{SLat}$.

Proof. The functor F is defined by letting $F(A) = \coprod_A I_{\mathbf{SLat}}$, the coproduct of A -many copies of the unit semilattice. The coproduct $S_1 \coprod S_2$ of two semilattices has as elements the formal joins $s_1 + s_2 + \dots + s_n$

with each s_i drawn from S_1 or S_2 , quotiented by the semilattice axioms together with the relations of S_1 and S_2 . The adjunction is then given by the following natural isomorphisms

$$\begin{aligned} \mathbf{SLat}(\coprod_A (I_{\mathbf{SLat}}), B) &\cong \prod_A (\mathbf{SLat}(I_{\mathbf{SLat}}, B)) \\ &\cong \mathbf{Set}(A, \mathbf{SLat}(I_{\mathbf{SLat}}, B)) \\ &\cong \mathbf{Set}(A, U(B)) \end{aligned}$$

The first isomorphism holds because the contravariant hom-functor sends colimits in its first argument to limits, so the coproduct becomes a product. The second is the isomorphism $\prod_A X \cong \mathbf{Set}(A, X)$. By the Yoneda lemma it suffices to give a bijection natural in Y :

$$\mathbf{Set}(Y, \prod_A X) \cong \prod_A \mathbf{Set}(Y, X) \cong \mathbf{Set}(\coprod_A Y, X) \cong \mathbf{Set}(Y \times A, X) \cong \mathbf{Set}(Y, \mathbf{Set}(A, X)),$$

using, in order, the universal property of the product, that of the coproduct (with $\coprod_A Y \cong Y \times A$), and currying. The last bijection is the definition of U . As all three bijections are natural in B , their composite exhibits $F \dashv U$. $\square$

Corollary A.12. The free functor $F: \mathbf{Set} \rightarrow \mathbf{SLat}$ is strong monoidal as a functor $(\mathbf{Set}, \times) \rightarrow (\mathbf{SLat}, \otimes)$.

Proof. We have $F(I_{\mathbf{Set}}) \cong I_{\mathbf{SLat}}$, and

$$\begin{aligned} F(A) \otimes F(B) &\cong (\coprod_A I_{\mathbf{SLat}}) \otimes (\coprod_B I_{\mathbf{SLat}}) \\ &\cong \coprod_A (I_{\mathbf{SLat}} \otimes \coprod_B I_{\mathbf{SLat}}) \\ &\cong \coprod_A (\coprod_B (I_{\mathbf{SLat}} \otimes I_{\mathbf{SLat}})) \\ &\cong \coprod_A (\coprod_B I_{\mathbf{SLat}}) \\ &\cong \coprod_{A \times B} I_{\mathbf{SLat}} \\ &\cong F(A \times B), \end{aligned}$$

naturally in A and B , where we used that $- \otimes X$ and $X \otimes -$ preserve colimits, being left adjoints by the symmetry and monoidal closedness of $\mathbf{SLat}$ (proposition A.8). $\square$

While a monoidal base suffices to define enriched categories, we assume that the base is symmetric monoidal closed, so that the standard category-theoretic machinery (such as the tensoring of enriched categories) is available in the enriched setting.

B String diagram rewrite theory

B.1 Hypergraphs

Proposition B.1 ([6]). $\mathbf{Hyp}(\Sigma)$ is equivalently the slice category $\mathbf{Hyp} \downarrow G_\Sigma$, where $\mathbf{Hyp}$ is the category of (unlabelled) hypergraphs — equivalently, the category of finite presheaves over the index category $\mathbf{I}$ whose objects are the pairs $(k, l) \in \mathbb{N} \times \mathbb{N}$ together with a distinguished object $\star$, with $k+l$ morphisms $\{i: \star \rightarrow (k, l) \mid 0 \leq i < k+l\}$ — and G_Σ is the hypergraph with a single vertex v and, for each generator $g: n \rightarrow m$ of Σ , one hyperedge e_g in which v occurs n times in $s(e_g)$ and m times in $t(e_g)$.

Intuitively, $\mathbf{I}$ presents a hypergraph by its vertices (the object $\star$) and its hyperedges sorted by arity (an object (k, l) is the sort of hyperedges with k sources and l targets), the $k+l$ maps picking out each source and target vertex. The hyperedges of G_Σ are then the available labels, and a morphism $H \rightarrow G_\Sigma$ in $\mathbf{Hyp}$ labels each hyperedge of H by the generator it is sent to, automatically matching arities. As a presheaf

category, **Hyp** has all finite colimits — the initial object is the empty hypergraph and coproducts are disjoint unions — and these are created by the forgetful functor out of the slice, so **Hyp**(Σ) inherits them.

The single vertex of G_Σ reflects that Σ here carries a single object type. For a many-sorted signature with a set of types (colours) C , one works instead with the category **Hyp** $_C$ of C -coloured hypergraphs — finite presheaves over the analogous index category **I** $_C$ in which the single vertex object $\star$ is replaced by one object $\star_c$ per colour $c \in C$, and the arity objects record the colour of each source and target — and takes G_Σ to have one vertex per colour, attaching each generator’s sources and targets to the vertices of the corresponding colours. The resulting slice **Hyp** $_C \downarrow G_\Sigma =: \mathbf{Hyp}_{C,\Sigma}$ then enforces the typing discipline of Definition 3.1 for free, since a labelling morphism must respect sources and targets and hence colours.

Proposition B.2. **HypI**(Σ) is a symmetric monoidal category.

Proof. Composition of two cospans $n \xrightarrow{f} \mathcal{F} \xleftarrow{g} m \xrightarrow{f'} \mathcal{G} \xleftarrow{g'} k$ is given by the pushout $n \xrightarrow{f} \mathcal{F} \sqcup_{g,f'} \mathcal{G} \xleftarrow{g'} k$, computed by identifying vertices in $\mathcal{F} \sqcup \mathcal{G}$ that share pre-image in m . The latter is a pushout in **Hyp**(Σ) and always exists since **Hyp**(Σ) has all finite colimits.

1. the tensor product given by coproduct in **Hyp**(Σ):

$$\begin{array}{ccc} n \xrightarrow{f} \mathcal{F} \xleftarrow{g} m & & \\ \otimes & ::= n \sqcup n' \xrightarrow{f \sqcup f'} \mathcal{F} \sqcup \mathcal{G} \xleftarrow{g \sqcup g'} m \sqcup m'; & \\ n' \xrightarrow{f'} \mathcal{G} \xleftarrow{g'} m' & & \end{array}$$

2. the monoidal unit is given by the initial object of **Hyp**(Σ);
3. symmetry is inherited from coproduct in **Hyp**(Σ) [16].

□

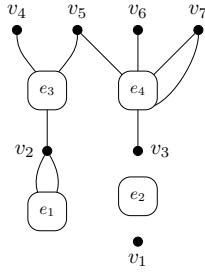
Definition B.3 (Category of MDA Hypergraphs with Interfaces [7]). We call a cospan $n \xrightarrow{f} \mathcal{G} \xleftarrow{g} m$ in **HypI**(Σ) monogamous if:

1. $\mathcal{G}$ is directed acyclic;
2. f and g are monomorphisms;
3. the in-degree and out-degree of every vertex is at most 1;
4. vertices with in-degree 0 are precisely the image of f ;
5. vertices with out-degree 0 are precisely the image of g .

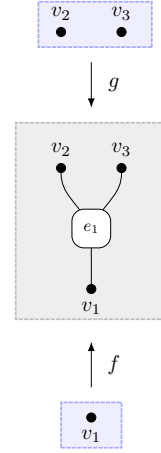
We denote by **MHypI**(Σ) the subcategory of **HypI**(Σ) of monogamous directed acyclic (MDA) cospans.

Figure 8b gives an example of an MDA cospan, which is such because it “looks like” the string diagram Υ . Analogously, we can see that Figure 8a (equipped with the interfaces choosing $\{v_1\}$ and $\{v_4, v_5, v_6, v_7\}$) fails to form an MDA cospan — it is not monogamous for many reasons: v_1 has out-degree 0, v_1, v_5, v_7 have in-degree 2, and v_3 has in-degree 0; additionally, it fails to be directed acyclic due to the path along e_4 from v_7 to itself¹¹.

¹¹Recall that loops are not part of the syntax of string diagrams for symmetric monoidal categories in the absence of a trace — a relaxation of monogamy to allow for this is explored by Ghica et al. [19].



(a) Hypergraph with 7 vertices and 4 hyperedges, all directed bottom-to-top; source/target vertices are ordered left-to-right.



(b) Hypergraph with interfaces

Figure 8: Example hypergraphs

B.2 DPOI-Rewriting for Hypergraphs with Interfaces

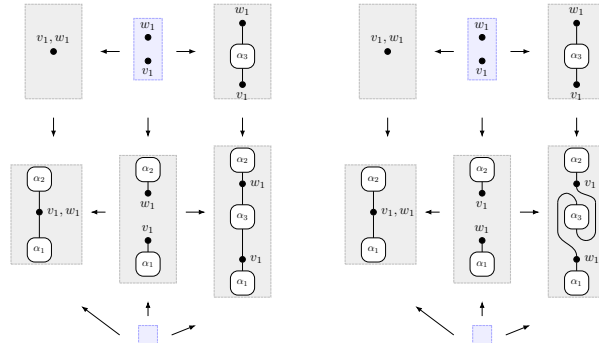
Definition B.4 (Boundary complement). For MDA cospans $i \xrightarrow{a_1} \mathcal{L} \xleftarrow{a_2} j$ and $n \xrightarrow{b_1} \mathcal{G} \xleftarrow{b_2} m$ and monomorphism $f: \mathcal{L} \rightarrow \mathcal{G}$, a pushout complement $i \sqcup j \rightarrow \mathcal{L}^\perp \rightarrow \mathcal{G}$ as depicted in the square below

$$\begin{array}{ccc}
 \mathcal{L} & \xleftarrow{[a_1, a_2]} & i \sqcup j \\
 f \downarrow & & \downarrow [c_1, c_2] \\
 \mathcal{G} & \xleftarrow{\quad} & \mathcal{L}^\perp \\
 & \nearrow [b_1, b_2] & \uparrow [d_1, d_2] \\
 & & n \sqcup m
 \end{array}$$

is called a boundary complement if $[c_1, c_2]$ is a monomorphism and there exist $d_1: n \rightarrow \mathcal{L}^\perp$ and $d_2: m \rightarrow \mathcal{L}^\perp$ making the above triangle commute and such that $n \sqcup j \xrightarrow{[d_1, c_2]} \mathcal{L}^\perp \xleftarrow{[d_2, c_1]} m \sqcup i$ is an MDA cospan.

Intuitively, requiring boundary complements ensures that inputs are glued exclusively to outputs and vice-versa in the two pushout squares.

Example B.5 (Example 29 [7]). Consider the following signature $\Sigma = \{\alpha_1: 0 \rightarrow 1, \alpha_2: 1 \rightarrow 0, \alpha_3: 1 \rightarrow 1\}$ and the rewrite rule $id \rightarrow \alpha_3$ for the example term $\alpha_1 \circ \alpha_2$. Without the boundary complement condition, we get, for example, the following two possible DPOI squares.



The second DPOI square does not satisfy the boundary complement condition as the input interface is mapped to the input vertex (and similarly for the output interface) and therefore, once pushout square is complete, the resulting cospan does not represent a valid SMC morphism as it requires wires to be able to go backwards, i.e. it requires compact closed structure. If we were to consider non-monic maps into $\mathcal{L}^\perp$ we would have more DPOI squares non of which would result in a valid morphism (due to in- or out- degrees of vertices being greater than 1).

Definition B.6 (Convex match). A subgraph $\mathcal{H}$ of a hypergraph $\mathcal{G}$ is a convex subgraph if for all $v_i, v_j \in V_{\mathcal{H}}$ every path from v_i to v_j is also in $\mathcal{H}$. A match $f: \mathcal{L} \rightarrow \mathcal{G}$ is a convex match if it is a monomorphism and its image is a convex subgraph of $\mathcal{G}$.

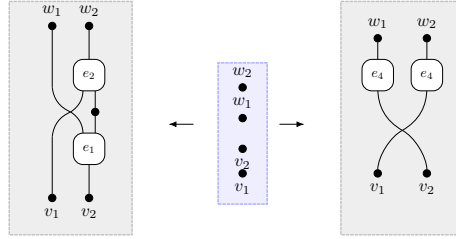
Definition B.7 (Convex DPOI rewriting). Let $\mathfrak{R}$ be a set of DPOI rewrite rules. Then, given $\mathcal{G} \leftarrow n \sqcup m$ and $\mathcal{H} \leftarrow n \sqcup m$ in $\mathbf{Hyp}(\Sigma)$, $\mathcal{G}$ rewrites convexly into $\mathcal{H}$ with interface $n \sqcup m$, and we write $(\mathcal{G} \leftarrow n \sqcup m) \Rightarrow_{\mathfrak{R}} (\mathcal{H} \leftarrow n \sqcup m)$, if

1. there exists a rule $\mathcal{L} \leftarrow i \sqcup j \rightarrow \mathcal{R}$ in $\mathfrak{R}$ and object $\mathcal{L}^\perp$ and cospan arrows $i \sqcup j \rightarrow \mathcal{L}^\perp \leftarrow n \sqcup m$ such that the DPOI diagram of Definition 3.4 commutes and its marked squares are pushouts;
2. $f: \mathcal{L} \rightarrow \mathcal{G}$ is a convex match;
3. $i \sqcup j \rightarrow \mathcal{L}^\perp \rightarrow \mathcal{H}$ is a boundary complement in the leftmost pushout.

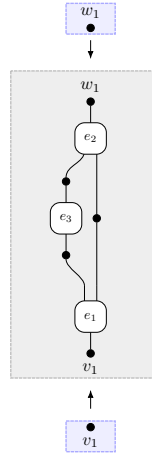
Example B.8 (Example 32 [7]). Consider the following signature $\Sigma\{e_1: 1 \rightarrow 2, e_2: 2 \rightarrow 1, e_3: 1 \rightarrow 1, e_4: 1 \rightarrow 1\}$ and the following rewrite rule

$$id \otimes e_1 \circ sym \otimes id \circ id \otimes e_2 \rightarrow sym \circ e_4 \otimes e_4$$

represented as the following span of hypergraphs



Now consider the cospan of hypergraphs corresponding to the following Σ -term $e_1 \circ e_3 \otimes id \circ e_2$.



Applying the rule to the cospan above results in the DPOI square from Figure 9. While the DPOI square is valid and the pushout complement satisfies the boundary complement condition, the match (highlighted in red) is not convex as there exists a path from the input of e_3 to its output that does not belong to the match itself. The consequence of this is that this rewrite step does not correspond to a term rewriting modulo SMC — there is no way to identify such a redex without at least traced monoidal structure. That is, there is no decomposition of $e_1 \circ e_3 \otimes id \circ e_2$ as $c_1 \circ id_n \otimes (id \otimes e_1 \circ sym \otimes id \circ id \otimes e_2) \circ c_2$. There is,

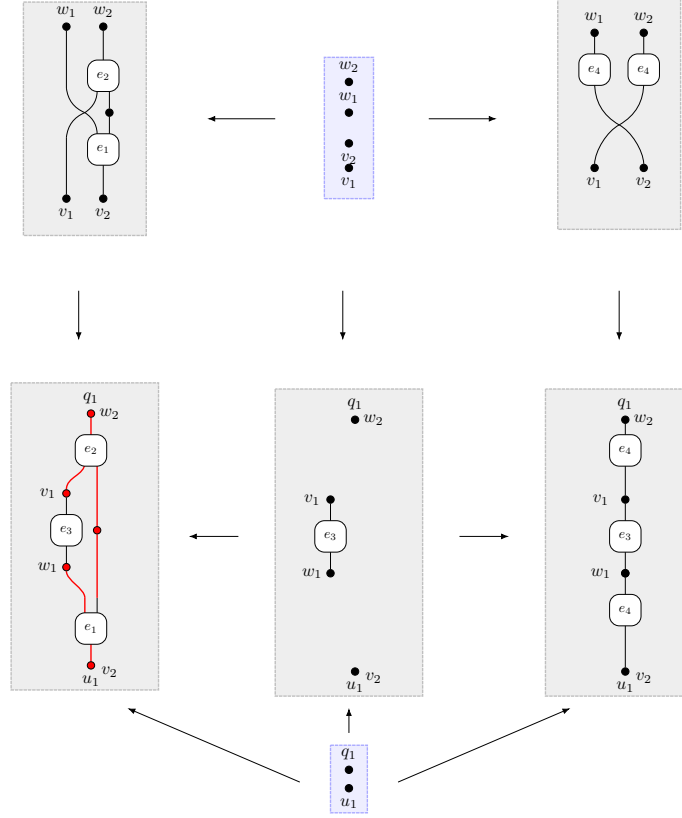


Figure 9: DPOI diagram for a non-convex match

however, a match modulo traced symmetric monoidal structure — $\text{tr}(e_3 \otimes id \circ e_1 \otimes id \circ \text{sym} \otimes id \circ id \otimes e_2)$. Requiring convex matches prevents such situations from arising.

C Pushout in $\text{EHyp}(\Sigma)$

In constructing the pushout the functional versions of e-hypergraph relations will be useful.

Remark C.1. $\prec$, $<$ and $\smile$ can be considered as (partial) functions defined on $V_{\mathcal{F}} \sqcup E_{\mathcal{F}}$, i.e., on the coproduct of vertices and edges. To make things well-typed, we will use corresponding coproduct injections $\iota_V : V_{\mathcal{F}} \rightarrow V_{\mathcal{F}} \sqcup E_{\mathcal{F}}$ and $\iota_E : E_{\mathcal{F}} \rightarrow V_{\mathcal{F}} \sqcup E_{\mathcal{F}}$ when passing either a vertex or an edge into these functions. For example, the immediate predecessor of a vertex x can be written functionally as $\prec_{\mathcal{F}}^{\mu}(\iota_{V_{\mathcal{F}}}(x))$.

Remark C.2. Using functional notation we can reformulate the notion of a homomorphism between two e-hypergraphs in the following way.

Definition C.3. A *homomorphism* $\phi : \mathcal{F} \rightarrow \mathcal{G}$ of e-hypergraphs $\mathcal{F}, \mathcal{G}$ is a pair of functions $\phi_V : V_{\mathcal{F}} \rightarrow V_{\mathcal{G}}$, $\phi_E : E_{\mathcal{F}} \rightarrow E_{\mathcal{G}}$ such that

1. ϕ is hypergraph homomorphism.
2. When x is not a top-level vertex such that $\prec(\iota_V(x))$ is defined,

$$\phi_E(\prec_{\mathcal{F}}^{\mu}(\iota_{V_{\mathcal{F}}}(x))) = \prec_{\mathcal{G}}^{\mu}(\phi_V \circ \iota_{V_{\mathcal{F}}}(x))$$

and

$$\phi_E(\prec_{\mathcal{F}}^{\mu}(\iota_{E_{\mathcal{F}}}(x))) = \prec_{\mathcal{G}}^{\mu}(\phi_E \circ \iota_{E_{\mathcal{F}}}(x))$$

when x is a not top-level edge.

3. When x is not a top-level vertex such that $<(\iota_V(x))$ is defined,

$$\phi_E(<_{\mathcal{F}}^{\mu}(\iota_{V_{\mathcal{F}}}(x))) = <_{\mathcal{G}}^{\mu}(\phi_V \circ \iota_{V_{\mathcal{G}}}(x))$$

and

$$\phi_E(<_{\mathcal{F}}^{\mu}(\iota_{E_{\mathcal{F}}}(x))) = <_{\mathcal{G}}^{\mu}(\phi_E \circ \iota_{E_{\mathcal{G}}}(x))$$

when x is a not top-level edge.

4. When $x \in E_{\mathcal{F}}$

$$[\phi_V \circ \iota_{V_{\mathcal{G}}}, \phi_E \circ \iota_{E_{\mathcal{G}}}]^*(\neg_{\mathcal{F}}(\iota_{E_{\mathcal{F}}}(x))) \subseteq \neg_{\mathcal{G}}(\phi_E \circ \iota_{E_{\mathcal{G}}}(x))$$

where $\phi_V \circ \iota_{V_{\mathcal{G}}}: V_{\mathcal{F}} \rightarrow V_{\mathcal{G}} \sqcup E_{\mathcal{G}}$, and similarly for $\phi_E \circ \iota_{E_{\mathcal{G}}}$ so that $[\phi_V \circ \iota_{V_{\mathcal{G}}}, \phi_E \circ \iota_{E_{\mathcal{G}}}] : V_{\mathcal{F}} \sqcup E_{\mathcal{F}} \rightarrow V_{\mathcal{G}} \sqcup E_{\mathcal{G}}$.

5. When $x \in V_{\mathcal{F}}$

$$[\phi_V \circ \iota_{V_{\mathcal{G}}}, \phi_E \circ \iota_{E_{\mathcal{G}}}]^*(\neg_{\mathcal{F}}(\iota_{V_{\mathcal{F}}}(x))) \subseteq \neg_{\mathcal{G}}(\phi_V \circ \iota_{V_{\mathcal{G}}}(x)).$$

Theorem C.4 (Existence of pushouts in **EHyp**(Σ)). Consider the following span in **EHyp**(Σ)

$$\begin{array}{ccc} Z & \xrightarrow{f} & X \\ g \downarrow & & \\ Y & & \end{array}$$

such that

1. Z is a *discrete* e-hypergraph.
2. $[f_V(v_i)] = [f_V(v_j)]$ and $[g_V(v_i)] = [g_V(v_j)]$ for all v_i, v_j in V_Z .
3. If $[f_V(v)] \neq \emptyset$ then $[g_V(v)] = \emptyset$ and if $[g_V(v)] \neq \emptyset$ then $[f_V(v)] = \emptyset$.
4. $\neg(f_V(v_i)) = \neg(f_V(v_j))$ and $\neg(g_V(v_i)) = \neg(g_V(v_j))$ for all v_i, v_j in V_Z .

then the pushout $X \sqcup_{f,g} Y$ exists.

Proof. We next explicitly construct a pushout. Consider the diagram below.

$$\begin{array}{ccccc} Z & \xrightarrow{f} & X & & \\ g \downarrow & & \downarrow \iota_1 / \sim_R & \searrow j_1 & \\ Y & \xrightarrow{\iota_2 / \sim_R} & X \sqcup Y / \sim_R & \xrightarrow{u} & Q \\ & \searrow j_2 & & & \end{array}$$

Then, the pushout of e-hypergraphs X and Y is computed in two steps. First, a coproduct of $X \sqcup Y$ is computed which is

$$X \sqcup Y = \{V_X \sqcup V_Y, E_X \sqcup E_Y, s_{X \sqcup Y}, t_{X \sqcup Y}, \neg_{X \sqcup Y}, <_{X \sqcup Y}^{\text{orange}}, <_{X \sqcup Y}^{\text{blue}}\}$$

where $s_{X \sqcup Y}: E_X \sqcup E_Y \rightarrow (V_X \sqcup V_Y)^*$ which can be defined as a copairing $[s'_X, s'_Y]: E_X \sqcup E_Y \rightarrow (V_X \sqcup V_Y)^*$ and $s'_X: E_X \rightarrow (V_X \sqcup V_Y)^*$, $s'_Y: E_Y \rightarrow (V_X \sqcup V_Y)^*$ defined as $s'_X = s_X \circ \iota_{1,V}^*$ and $s'_Y = s_Y \circ \iota_{2,V}^*$ where ι_1, ι_2 are corresponding coproduct injections, similarly for $t_{X \sqcup Y}, <_{X \sqcup Y}^{\text{orange}}, <_{X \sqcup Y}^{\text{blue}}, \neg_{X \sqcup Y}$. We will omit labels as they are

irrelevant to pushout construction. Consider relations

$$\begin{aligned}
S_V = & \{ (x_i, y_j) \in (V_X \sqcup V_Y) \times (V_X \sqcup V_Y) \mid \\
& \exists z \in V_Z . x_i = f_V \circ \iota_{1,V}(z) \text{ and } y_j = g_V \circ \iota_{2,V}(z) \\
& \text{where } x_i \in V_X \text{ and } y_j \in V_Y \} \\
& \cup \\
& \{ (y_j, x_i) \in (V_X \sqcup V_Y) \times (V_X \sqcup V_Y) \mid \\
& \exists z \in V_Z . x_i = f_V \circ \iota_{1,V}(z) \text{ and } y_j = g_V \circ \iota_{2,V}(z) \\
& \text{where } x_i \in V_X \text{ and } y_j \in V_Y \} \\
& \cup \\
& \{ (x, x) \text{ where } x \in V_X \sqcup V_Y \}
\end{aligned}$$

$$S_E = \{ (x, x) \text{ where } x \in E_X \sqcup E_Y \}$$

and let relations R_V, R_E be their transitive closures respectively.

We then quotient the set vertices and edges in $X \sqcup Y$ by R_V and R_E respectively.

$$\begin{aligned}
X \sqcup Y / \sim_{(R_V, R_E)} = & (V_X \sqcup V_Y / \sim_{(R_V, R_E)}, \\
& E_X \sqcup E_Y / \sim_{(R_V, R_E)}, \\
& s_{X \sqcup Y} / \sim_{(R_V, R_E)}, \\
& t_{X \sqcup Y} / \sim_{(R_V, R_E)}, \\
& \smile_{X \sqcup Y} / \sim_{(R_V, R_E)}, \\
& \textcolor{orange}{<}_{X \sqcup Y} / \sim_{(R_V, R_E)} \\
& \textcolor{blue}{<}_{X \sqcup Y} / \sim_{(R_V, R_E)})
\end{aligned}$$

We will refer to $\sim_{(R_V, R_E)}$ just as $\sim$, *e.g.*, by writing $V_X \sqcup V_Y / \sim$ and concrete relation will be clear from the context. Where necessary we will refer to S_V and S_R as $\sim_S$. We have

$$s_{X \sqcup Y} / \sim : E_X \sqcup E_Y / \sim \rightarrow (V_X \sqcup V_Y / \sim)^*$$

and there is an obvious surjective function $[\]_V : (V_X \sqcup V_Y) \rightarrow (V_X \sqcup V_Y / \sim)$ that maps elements to their equivalence classes and $[\]_V^*$ is its extension to sequences. There is also $[\] : E_X \sqcup E_Y \rightarrow E_X \sqcup E_Y / \sim$ and we will omit subscripts as the correct type will be clear from the argument. We then define

$$s_{X \sqcup Y} / \sim ([e]) = s_{X \sqcup Y} \circ [\]^*(e) = [s_{X \sqcup Y}(e)]^*$$

We will also use subscripts when it is important to tell if an element of $E_X \sqcup E_Y$ has a pre-image in either E_X or E_Y by writing e_x or e_y . Similarly, we will write v_x to refer to a vertex with a pre-image in V_X . Likewise,

$$t_{X \sqcup Y} / \sim ([e]) = [t_{X \sqcup Y}(e)]^*$$

These definitions make $([\]_V, [\]_E)$ automatically a homomorphism with respect to source and target maps. These maps are also automatically well-defined functions since $[e_1] = [e_2]$ implies $e_1 = e_2$ since the mappings for edges are mono.

Recall that $<$ is essentially a transitive closure of $<^\mu$ and hence we can first define

$$\textcolor{orange}{<}_{X \sqcup Y}^\mu / \sim (t_{E_X \sqcup E_Y} / \sim [e])$$

and

$$\textcolor{orange}{<}_{X \sqcup Y}^\mu / \sim (t_{V_X \sqcup V_Y} / \sim [u])$$

where ι_* are injections into $V_X \sqcup V_Y / \sim \sqcup E_X \sqcup E_Y / \sim$. We will consider several cases. First, assume that u is an element of $V_{X \sqcup Y}$ and

1. If there exists v such that $\prec_{X \sqcup Y}^\mu(\iota_{V_X \sqcup V_Y}(v))$ is defined and $u \sim v$ we let

$$\prec_{X \sqcup Y/\sim}^\mu(\iota_{V_X \sqcup V_Y/\sim}([u])) = [\prec_{X \sqcup Y}^\mu(\iota_{V_X \sqcup V_Y}(v))]$$
2. u has no pre-image in V_Z and $\prec_{X \sqcup Y}^\mu(\iota_{V_X \sqcup V_Y}(u))$ is undefined (*i.e.*, $[u] = \emptyset$). If there exists v such that $\prec_{X \sqcup Y}^\mu(\iota_{V_X \sqcup V_Y}(v)) = e'$ and such that there is an *undirected* path from $[u]$ to $[v]$, then we define

$$\prec_{X \sqcup Y/\sim}^\mu(\iota_{V_X \sqcup V_Y/\sim}([u])) = \prec_{X \sqcup Y/\sim}^\mu(\iota_{V_X \sqcup V_Y/\sim}([v]))$$

Otherwise, we leave $\prec_{X \sqcup Y/\sim}^\mu(\iota_{V_X \sqcup V_Y/\sim}([u]))$ undefined. Now, assume that e is an element of $E_{X \sqcup Y}$. Consider two cases.

1. $\prec_{X \sqcup Y}^\mu(\iota_{E_X \sqcup E_Y}(e)) = e'$. Then we define

$$\prec_{X \sqcup Y/\sim}^\mu(\iota_{E_X \sqcup E_Y/\sim}([e])) = [\prec_{X \sqcup Y}^\mu(\iota_{E_X \sqcup E_Y}(e))]$$
2. $[e] = \emptyset$ and there exists v such that $\prec_{X \sqcup Y}^\mu(\iota_{V_X \sqcup V_Y}(v)) = e'$ and such that there is an undirected path from $[e]$ to $[v]$, we define

$$\prec_{X \sqcup Y/\sim}^\mu(\iota_{E_X \sqcup E_Y/\sim}([e])) = \prec_{X \sqcup Y/\sim}^\mu(\iota_{V_X \sqcup V_Y/\sim}([v]))$$

Otherwise we leave $\prec_{X \sqcup Y/\sim}^\mu(\iota_{E_X \sqcup E_Y/\sim}([e]))$ undefined. Clearly, all the cases above are disjoint. We similarly define $\prec_{X \sqcup Y/\sim}^\mu$.

Let's now define $\smile_{X \sqcup Y/\sim}$. We can consider the consistency relation from the coproduct as a function

$$\smile_{X \sqcup Y} : (V_X \sqcup V_Y) \sqcup (E_X \sqcup E_Y) \rightarrow 2^{(V_X \sqcup V_Y) \sqcup (E_X \sqcup E_Y)}$$

Quotienting the values of the function gives us

$$\smile'_{X \sqcup Y} : V_X \sqcup V_Y \sqcup E_X \sqcup E_Y \rightarrow 2^{(V_X \sqcup V_Y/\sim \sqcup (E_X \sqcup E_Y)/\sim)}$$

which is essentially $[[\Box_V \circ \iota_{V_X \sqcup V_Y/\sim}, [\Box_E \circ \iota_{E_X \sqcup E_Y/\sim}]]^*$ (a copairing extended to sequences that we will further denote as $[[\Box_V], [\Box_E]]$) applied to the return value of $\smile_{X \sqcup Y}$. We then first define an auxiliary relation $\smile^\#$ similarly to how $\prec_{X \sqcup Y/\sim}^\mu$ was defined. We begin with defining $\smile^\#_{X \sqcup Y/\sim}$ for vertices.

1. If there exists v such that $\smile_{X \sqcup Y}(\iota_{V_X \sqcup V_Y}(v)) \neq \emptyset$ and $u \sim v$, we let

$$\smile^\#_{X \sqcup Y/\sim}(\iota_{V_X \sqcup V_Y/\sim}([u])) =$$

$$=$$

$$[[\Box_V], [\Box_E]]^*(\smile_{X \sqcup Y}(\iota_{V_X \sqcup V_Y}(v)))$$
2. u has no pre-image in V_Z and $\smile_{X \sqcup Y}(\iota_{V_X \sqcup V_Y}(u)) = \emptyset$ and there exists v such that $\smile_{X \sqcup Y}(\iota_{V_X \sqcup V_Y}(v)) \neq \emptyset$ and such that there is an undirected path from $[u]$ to $[v]$. Then we let

$$\smile^\#_{X \sqcup Y/\sim}(\iota_{V_X \sqcup V_Y/\sim}([u])) = \smile^\#_{X \sqcup Y/\sim}(\iota_{V_X \sqcup V_Y/\sim}([v]))$$

Next we define $\smile^\#_{X \sqcup Y/\sim}$ for edges.

1. If $\smile_{X \sqcup Y}(\iota_{E_X \sqcup E_Y}(e)) \neq \emptyset$, then

$$\smile^\#_{X \sqcup Y/\sim}(\iota_{E_X \sqcup E_Y/\sim}([e]_E)) = [[\Box_V], [\Box_E]]^*(\smile_{X \sqcup Y}(\iota_{E_X \sqcup E_Y}(e)))$$
2. $\smile_{X \sqcup Y}(\iota_{E_X \sqcup E_Y}(e)) = \emptyset$ and there exists v such that $\smile_{X \sqcup Y}(\iota_{V_X \sqcup V_Y}(v)) \neq \emptyset$ and such that there is an undirected path from $[e]$ to $[v]$ then

$$\smile^\#_{X \sqcup Y/\sim}(\iota_{E_X \sqcup E_Y/\sim}([e]_E)) = \smile^\#_{X \sqcup Y/\sim}(\iota_{V_X \sqcup V_Y/\sim}([v]))$$

The well-definedness of this construction follows by the same argument as the well-definedness of $\prec_{X \sqcup Y/\sim}^\mu$. Then we define

$$\smile_{X \sqcup Y/\sim}(\iota_{V_X \sqcup V_Y/\sim}([v]))$$

and

$$\smile_{X \sqcup Y/\sim}(\iota_{E_X \sqcup E_Y/\sim}([e]))$$

as closures of $\smile^\#_{X \sqcup Y/\sim}$ as below.

$$(\smile^\#_{X \sqcup Y/\sim}(\iota_{\sim}([x])))^c$$

where c denotes a closure and $\iota_{\sim}$ is $\iota_{V_X \sqcup V_Y/\sim}$ or $\iota_{E_X \sqcup E_Y/\sim}$ depending on whether $[x_i]$ comes from $V_X \sqcup V_Y$ or $E_X \sqcup E_Y$, is the smallest set such that

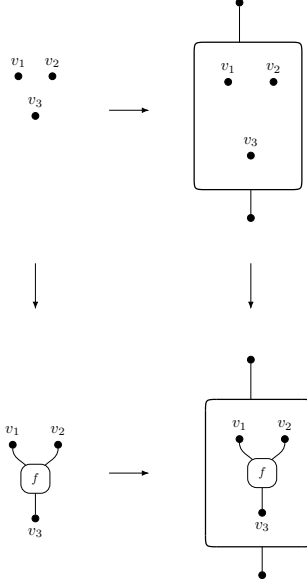


Figure 10: Pushout in $\mathbf{EHyp}(\Sigma)$

- $[x] \in (\neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x])))^c$ if $\neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x])) \neq \emptyset$
- if $[y] \in \neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x]))$ then

$$[x] \in (\neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([y])))^c$$

- for any sequence $([x_1], \dots, [x_n])$ such that

$$[x_i] \in \neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x_{i \sqcup 1}]))$$

or

$$[x_{i \sqcup 1}] \in \neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x_i]))$$

for $i < n$ both

$$[x_1] \in (\neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x_n])))^c$$

and

$$[x_n] \in (\neg_{x \sqcup y / \sim}^{\#}(\iota_{*/\sim}([x_1])))^c$$

This construction is analogous to how a pushout was constructed by Tiurin et al. [33] and we refer the reader for more detailed proofs there. The only addition is $\neg_{x \sqcup y / \sim}^{\mu}$ that is constructed analogously to $\neg_{x \sqcup y / \sim}^{\mu}$. $\square$

The construction can be illustrated with the example in Figure 10. Vertices and edges that are being identified as they have a common preimage inherit the relations from the vertex or edge that has it

defined either directly or via a path. In particular, a coproduct of two feet contains $\begin{matrix} v_1 & v_2 & v_3 & v'_1 & v'_2 & v'_3 \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{matrix}$ such that $\neg_{x \sqcup y}$ is only defined for v_1, v_2, v_3 . Therefore, we assign $\neg_{x \sqcup y / \sim}(\iota_{v_x \sqcup v_y / \sim}[v'_1]) = [\neg_{x \sqcup y}(\iota_{v_x \sqcup v_y}(v_1))]$ and so on. This motivates the requirement for conditions (2), (3), and (4) of Theorem C.4, otherwise such inheritance would be ambiguous.

D Interpretation of Σ -terms as cospans of hypergraphs

The base cases for interpretation function $[-]: \mathbf{S}(\Sigma) \rightarrow \mathbf{MHypI}(\Sigma)$ are given in Figure 11. Then it is defined inductively by letting $[f \otimes g] := [f] \otimes [g]$ and $[f \circ g] := [f] \circ [g]$ and by freeness induces the

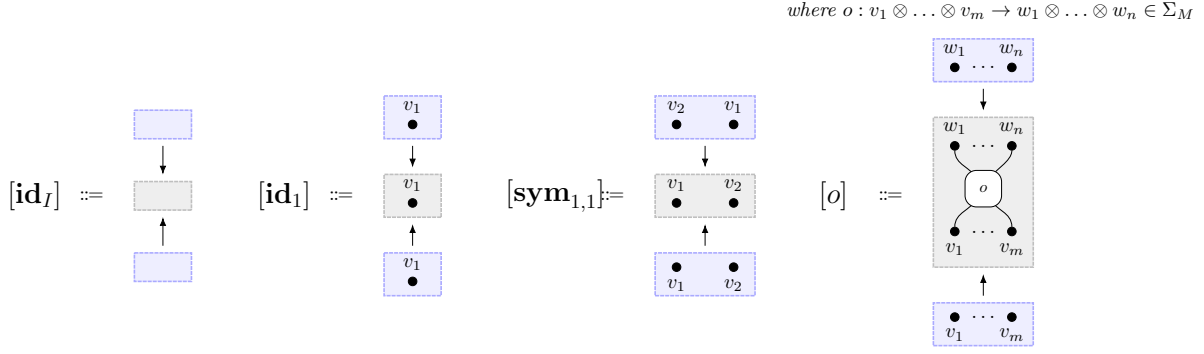
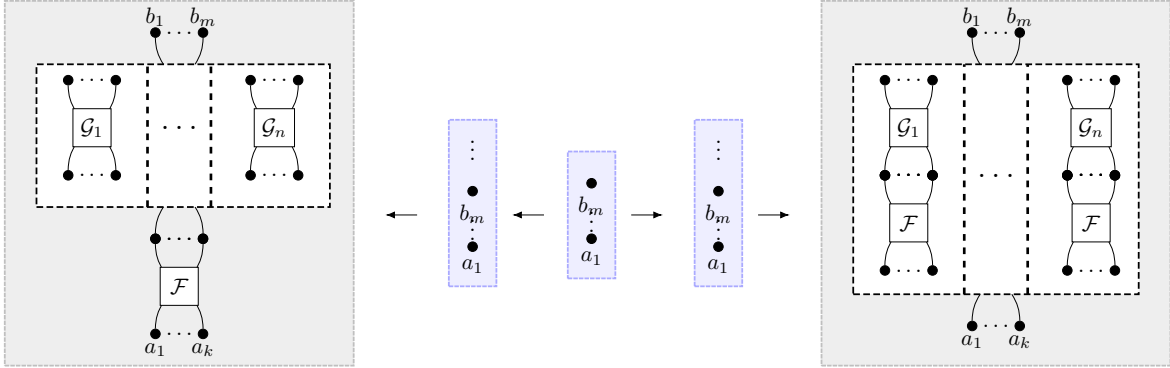
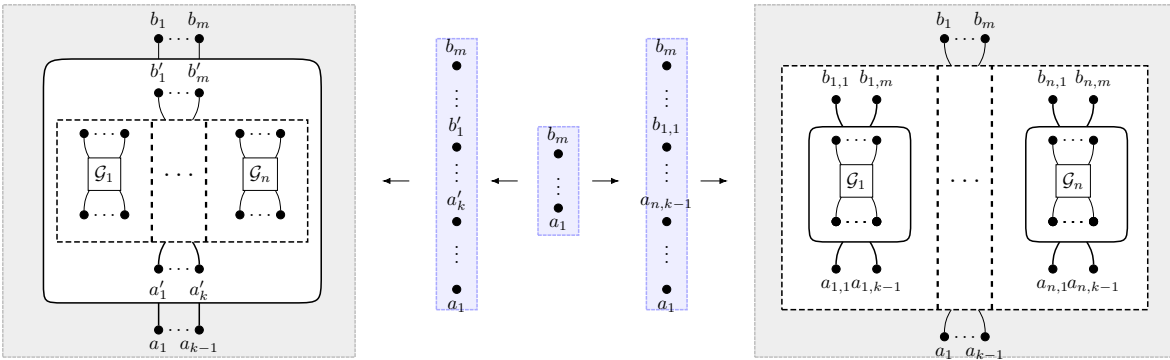


Figure 11: Base cases for $[-] : \mathbf{SMT}(\Sigma) \rightarrow \mathbf{MHypI}(\Sigma)$

corresponding functor. We obtain the interpretation function $\llbracket - \rrbracket : \Sigma^+ \rightarrow \mathbf{MCEHypI}(\Sigma)$ by combining the above with the interpretations of $\llbracket \mathbf{ev}_{A,B} \rrbracket$ and $\llbracket \Lambda_{A,B,C} \rrbracket$ shown in Figure 12 and by interpreting $\llbracket f+g \rrbracket$ as shown in Figure 13. The rest again follows by induction and freeness. Note that unlike $[-]$, $\llbracket - \rrbracket$ does not get promoted to a functor $\mathbf{CS}^+(\Sigma) \rightarrow \mathbf{MCEHypI}(\Sigma)$; this is because target of the to-be functor is not a *closed* symmetric monoidal **SLat**-category, although we will turn it into a symmetric monoidal **SLat**-category. This is achieved by quotienting morphisms of $\mathbf{MCEHypI}(\Sigma)$ by EDPOI rewrite rules that arise from the interpretation of **SLat** SMC axioms. For example, the EDPOI rule corresponding to the distributivity of $(\circ)$ over $+$ is given by the following rewrite rule.



Additionally, we quotient by the distributivity law (Equation 1). The rewrite (schema) rule corresponding to it looks as follows.



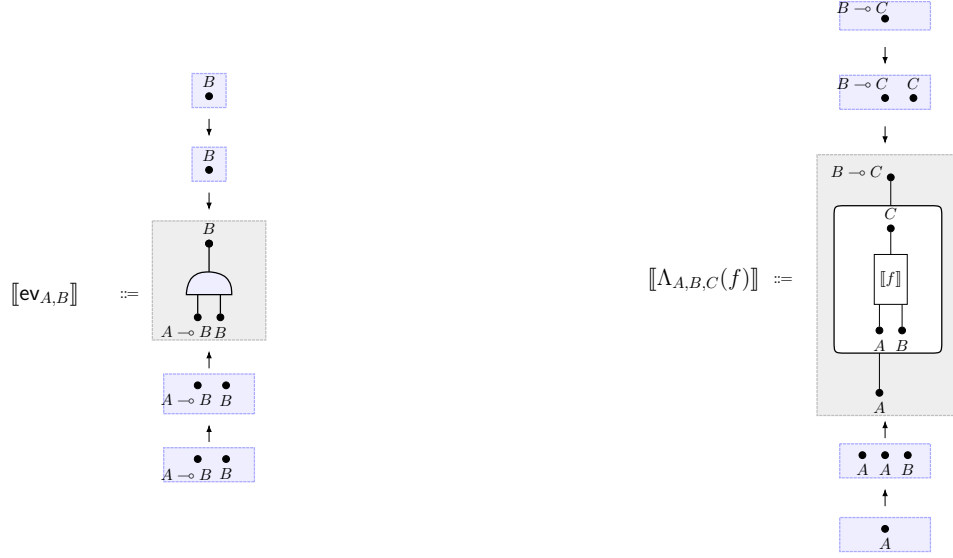


Figure 12: $\llbracket \text{ev} \rrbracket$ (left) and $\llbracket \Lambda(f) \rrbracket$ (right)

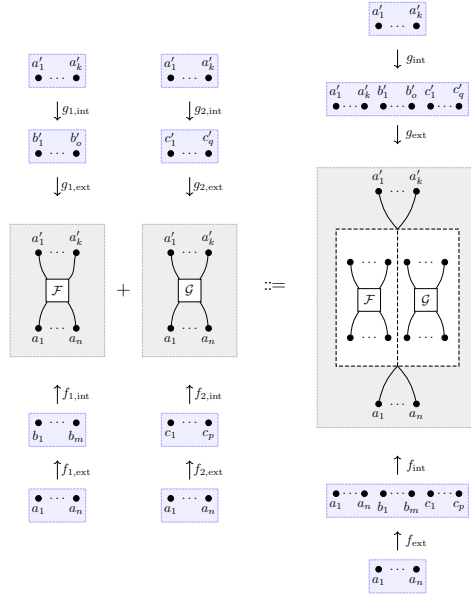


Figure 13: $+$ of two morphisms in $\mathbf{MCEHypI}(\Sigma)$

The collection of these rules is denoted $\mathcal{S}$ and defines the corresponding category $\mathbf{MCEHypI}(\Sigma)/\mathcal{S}$ where morphisms (cospans) are identified if they can be rewritten into each other.

E EDPOI rewriting

Consider an example of EDPOI rewriting in Figure 14 as defined in Definition 5.4. The rule in the top half of the diagram is an instance of a β -reduction rule as defined for string diagrams [18]. Note that the match satisfies all the conditions of the boundary complement as all vertices from the interface have the same parent. First, to compute the boundary complement, we remove from the source e-hypergraph

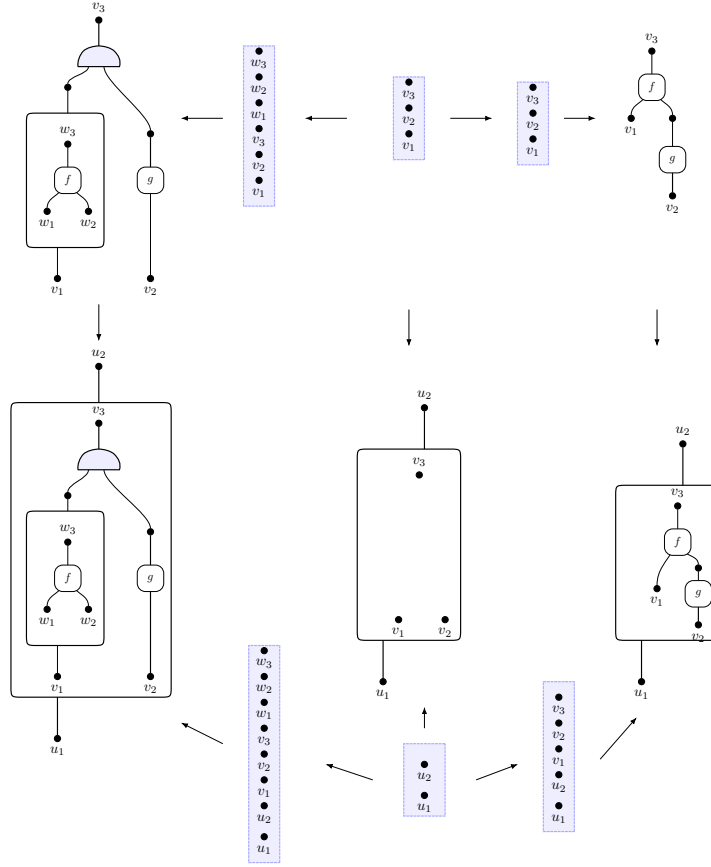


Figure 14: β -rule EDPOI example.

everything that has no pre-image in the interface above. The mda-cospan for the complement then will be

$$[u_1] \rightarrow [u_1, v_1, v_2] \sqcup [v_3] \rightarrow \mathcal{L}^\perp \leftarrow [v_3] \sqcup [v_1, v_2] \leftarrow [u_2] .$$

Note how we removed strict internal interfaces of $\mathcal{L}$ from internal interfaces of $\mathcal{G}$, yielding $[u_1, v_1, v_2]$ and similarly for output interfaces. After computing the pushout for the second half of the diagram we add strict internal interfaces of $\mathcal{R}$ but since there are none, we are done as per condition (2) of Definition 5.4.

F Modelling λ_{sub} -calculus in slotted e-graphs

```
pub enum Lambda {
  Lam(Bind<AppliedId>) = "lam",
  App(AppliedId, AppliedId) = "app",
  // subst lx t u := t[x / u]
  Subst(Bind<AppliedId>, AppliedId) = "subst",
  Var(Slot) = "var",
}
```

Listing 1: λ_{sub} syntax definition in slotted e-graphs

```

fn beta() -> Rewrite<Lambda> {
  let pat = "(app (lam $1 ?b) ?t)";
  let outputpat = "(subst $1 ?b ?t)";
  Rewrite::new("beta", pat, outputpat)
}

```

Listing 2: λ_{sub} $d\beta$ -rule

Recall the grammar, the reduction rules and the equational theory of λ_{sub} from Section 7. These can be directly encoded in the framework of slotted e-graphs of Schneider et al. [29] as can be seen in Listing 1, Listing 2, Listing 4, Listing 3, and Listing 5. We use **"subst"** to denote explicit substitution and use the built-in substitution mechanism provided by slotted e-graphs to encode $C[x][x/u] \rightarrow C[u][x/u]$ rules. Notably, since equivalence is symmetric, the corresponding rules have two versions — *left* and *right*. We check for the side conditions using conditional rewrites provided by the slotted e-graphs framework.

To show the effect of those particular sets of rewrites when performing equality saturation for λ_{sub} we apply certain subsets of these rewrites to a simple term that encodes 2+2 using Church encoding. When doing so, we keep track of the number of e-nodes in the saturated e-graph as well as the number of times each rewrite was applied during saturation (we count only the applications that change the e-graph). We also note whether a particular rewrite is absorbed by using string-diagrammatic (e-hypergraph) representation of λ_{sub} -terms. The idea is to show how much workload can be absorbed by using the above-mentioned representation that represents certain equivalences on the nose. Table 1 shows the number of e-nodes after saturation with different sets of rewrite rules associated with λ_{sub} enabled. **lambda_subst** is the distributivity of substitution over lambda abstraction (encoded using two rewrite rules) which is the middle equation of graphical equivalence from Section 7, **subst_comm** is the commutativity of substitutions which corresponds to the first equation and **app_subst** is the third equation. We can see that enabling all those rules — that is truly performing rewriting modulo graphical equivalence — significantly increases the number of e-nodes in the e-graph. Since **subst_comm** and **app_subst** are absorbed by the e-hypergraph representation we theoretically can get away with fewer nodes if we used this representation.

Another metric that we use is the number of times each rewrite was applied during saturation which is shown in Table 2. *gc* rule is encoded using **subst_no_var** rewrite rule. We can see that the rules that are absorbed by the e-hypergraph representation (highlighted in grey) comprise more than half of all the rewrite applications during saturation when all the rules are enabled. So again, using a more suitable representation can significantly reduce the workload of the equality saturation in this case.

Remark F.1. Even though *gc* rule is not absorbed by the e-hypergraph representation (despite being a part of string diagram equivalence), we can argue that this rule can be disregarded when using e-hypergraphs. Consider a substitution $y[x/v]$ which corresponds to the following composition in the internal language

$$\frac{\Gamma_1, y:T, \Gamma_2 \vdash v:T' \quad \Gamma_1, x:T', y:T, \Gamma_2 \vdash y:T}{\Gamma_1, y:T, \Gamma_2 \vdash y[x/v]:T \equiv y}$$

This composition is represented using the string diagram in Figure 15. While string diagrammatically this composition equals the string diagram for just y since we are working modulo the equations for copy and delete, they are not isomorphic as e-hypergraphs. However, since the output of v gets deleted anyway, we can ignore any rewrites that involve v and its inputs. The deletion morphisms for Γ_1, Γ_2 that follow the copy can be absorbed by the e-hypergraph representation by following the approach of Milosavljević et al. [25].

```

fn subst_no_var() -> Rewrite<Lambda> {
  let pat = "(subst $x ?t1 ?t2)";
  let outpat = "?t1";
  Rewrite::new_if("subst_no_var", pat, outpat, |subst, _ | {
    !subst["t1"].slots().contains(&Slot::named("x"))
  })
}

```

Listing 3: λ_{sub} gc-rule

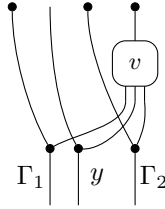


Figure 15: String diagram for $y[x/v]$

Set of equations	$d\beta + ls + gc$	$d\beta + ls + gc$ + lambda_subst	$d\beta + ls + gc$ + lambda_subst + subst_comm	$d\beta + ls + gc$ + lambda_subst + subst_comm + subst_app
# of e-nodes	88	185	244	846

Table 1: Number of e-nodes after saturating the e-graph for 2+2 with different sets of rewrite rules for λ_{sub}

Rewrite rule	# of applications ($d\beta + ls + gc$)	# of applications ($d\beta + ls + gc$ + lambda_subst)	# of applications ($d\beta + ls + gc$ + lambda_subst + subst_comm)	# of applications ($d\beta + ls + gc$ + lambda_subst + subst_comm + subst_app)	Absorbed?
beta	14	57	49	80	No
subst_no_var	35	113	118	197	No
ls_rule_*	48	141	140	302	Yes
subst_lambda	—	76	109	336	No
subst_comm	—	—	104	321	Yes
subst_app	—	—	—	263	Yes

Table 2: Number of times each rewrite rule was applied during saturation of the e-graph for 2+2

```

fn ls_rule_var() -> Rewrite<Lambda> {
  let pat = "(subst $x (var $x) ?t)";
  let outputpat = "(subst $x (var $x)[(var $x) := ?t] ?t)";
  Rewrite::new("ls_var", pat, outputpat)
}

fn ls_rule_app_l() -> Rewrite<Lambda> {
  let pat = "(subst $x (app ?u ?v) ?t)";
  let outputpat = "(subst $x (app ?u[(var $x) := ?t] ?v) ?t)";
  Rewrite::new_if("ls_app_l", pat, outputpat, |subst, _| {
    subst["u"].slots().contains(&Slot::named("x"))
    && !subst["v"].slots().contains(&Slot::named("x"))
  })
}

fn ls_rule_app_r() -> Rewrite<Lambda> {
  let pat = "(subst $x (app ?u ?v) ?t)";
  let outputpat = "(subst $x (app ?u ?v[(var $x) := ?t]) ?t)";
  Rewrite::new_if("ls_app_r", pat, outputpat, |subst, _| {
    !subst["u"].slots().contains(&Slot::named("x"))
    && subst["v"].slots().contains(&Slot::named("x"))
  })
}

fn ls_rule_lam() -> Rewrite<Lambda> {
  let pat = "(subst $x (lam $y ?b) ?t)";
  let outputpat = "(subst $x (lam $y ?b[(var $x) := ?t]) ?t)";
  Rewrite::new_if("ls_lam", pat, outputpat, |subst, _| {
    subst["b"].slots().contains(&Slot::named("x"))
  })
}

fn ls_rule_subst_l() -> Rewrite<Lambda> {
  let pat = "(subst $x (subst $y ?u ?t1) ?t2)";
  let outputpat = "(subst $x (subst $y ?u ?t1[(var $x) := ?t2]) ?t2)";
  Rewrite::new_if("ls_subst_l", pat, outputpat, |subst, _| {
    subst["t1"].slots().contains(&Slot::named("x"))
  })
}

fn ls_rule_subst_r() -> Rewrite<Lambda> {
  let pat = "(subst $x (subst $y ?u ?t1) ?t2)";
  let outputpat = "(subst $x (subst $y ?u[(var $x) := ?t2] ?t1) ?t2)";
  Rewrite::new_if("ls_subst_r", pat, outputpat, |subst, _| {
    subst["u"].slots().contains(&Slot::named("x"))
  })
}

```

Listing 4: λ_{sub} *ls*-rules

```

fn subst_comm() -> Rewrite<Lambda> {
  let pat = "(subst $y (subst $x ?u ?t1) ?t2)";
  let outpat = "(subst $x (subst $y ?u ?t2) ?t1)";
  Rewrite::new_if("subst_comm", pat, outpat, |subst, _| {
    !subst["t2"].slots().contains(&Slot::named("x"))
    && !subst["t1"].slots().contains(&Slot::named("y"))
  })
}

fn lambda_subst_l() -> Rewrite<Lambda> {
  let pat = "(subst $x (lam $y ?b) ?t)";
  let outpat = "(lam $y (subst $x ?b ?t))";
  Rewrite::new("lambda_subst_l", pat, outpat)
}

fn lambda_subst_r() -> Rewrite<Lambda> {
  let pat = "(lam $y (subst $x ?b ?t))";
  let outpat = "(subst $x (lam $y ?b) ?t)";
  Rewrite::new("lambda_subst_r", pat, outpat, |subst, _| {
    !subst["t"].slots().contains(&Slot::named("y"))
  })
}

fn subst_app_l() -> Rewrite<Lambda> {
  let pat = "(subst $x (app ?t1 ?t2) ?t3)";
  let outpat = "(app (subst $x ?t1 ?t3) ?t2)";
  Rewrite::new_if("subst_app_2", pat, outpat, |subst, _| {
    !subst["t2"].slots().contains(&Slot::named("x"))
  })
}

fn subst_app_r() -> Rewrite<Lambda> {
  let pat = "(app (subst $x ?t1 ?t3) ?t2)";
  let outpat = "(subst $x (app ?t1 ?t2) ?t3)";
  Rewrite::new_if("subst_app_2_2", pat, outpat, |subst, _| {
    !subst["t2"].slots().contains(&Slot::named("x"))
  })
}

```

Listing 5: Graphical equivalence rules for λ_{sub}