Hypergraphs with Binding

Anna Matsui[1] Koko Muroya[2]

[1]Johns Hopkins University, USA

[2]Ochanomizu University, Japan

TERMGRAPH
Lisbon, July 19, 2026

[1]Supported by ARO W911NF-20-1-0082
[2]Supported by JSPS, KAKENHI Project No. 22K17850, Japan

Outline

Motivation

We want to think about program comparison in terms of *improvement* — i.e., one program taking less time to execute than another one, while having the same outputs. For that, we also want to have a robust and automabale proof methodology.
Our goal is to build a combinatorial representation of programs that is as expressive as higher order term rewriting systems, and with a solid formalisation of rewriting for them.

	[2]	[5, 4]	this work
combinatorial representation of programs	✓	×	✓
solid formalisation of rewriting	×	✓	✓
proof methodology of improvement	✓	✓	×
—robustness of proof methodology	✓	×	–
—automation of proof methodology	×	✓	–

The development of an actual proof methodology is delegated to future work.

Motivational Example

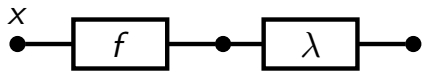
Here is an example of what we want to achieve in this work.
Consider a rewrite rule:
$(\lambda x.M[x])V \to M[V]$, with a substitution $V \mapsto \lambda x.t$ for some t.
We want to be able to express everything in this example in some sort of a graph rewriting framerwork.
For that, we would need to have the following:

- binding of x in $M[x]$;
- metavariable M in $M[x]$, that can take its own arguments;
- substitution of V in $M[V]$;
- rewrite rule, together with a context allowing its execution;

The talk will cover these points in this order, and finish with some more examples!

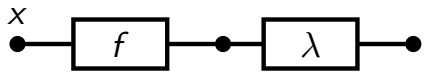
Examples

Consider an even simpler example: $\lambda x.f(x)$ for some f. We can try representing it as a hypergraph with two edges, λ and f:



Examples

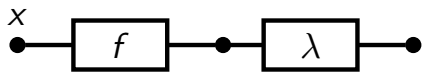
Consider an even simpler example: $\lambda x.f(x)$ for some f. We can try representing it as a hypergraph with two edges, λ and f:



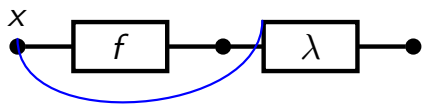
It is not clear from here that x is bound,

Examples

Consider an even simpler example: $\lambda x.f(x)$ for some f. We can try representing it as a hypergraph with two edges, λ and f:

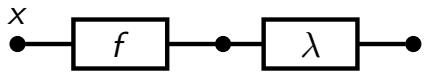


It is not clear from here that x is bound, so we add an extra connection to represent the binding relation:

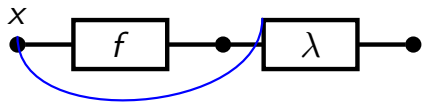


Examples

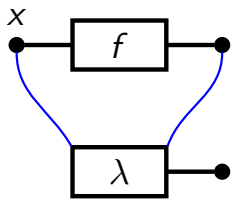
Consider an even simpler example: $\lambda x.f(x)$ for some f. We can try representing it as a hypergraph with two edges, λ and f:



It is not clear from here that x is bound, so we add an extra connection to represent the binding relation:



Or, to visually show that $f(x)$ is one layer more on the "inside" than λ, we can draw it like this:



Hypergraphs with Binding: Idea

To extend from a hypergraph to a hypergraph with binding, we add to the regular lists of targets and sources, a list of vertices for each element of a source list.

Hypergraphs with Binding: Idea

To extend from a hypergraph to a hypergraph with binding, we add to the regular lists of targets and sources, a list of vertices for each element of a source list.
Together, this data is given by two functions (plus the standard target function):

- $s : E \to V^*$ as before, and
- $b : E \to (V^*)^*$,

where for each edge $e \in E$ the length of the lists $s(e)$ and $b(e)$ is the same.
Or as one function

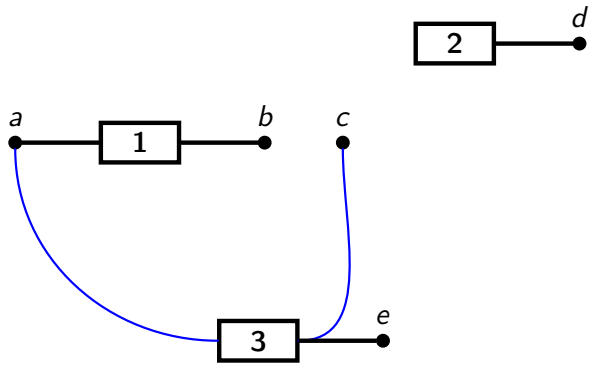
$$(s, b) : E \to (V \times V^*)^*,$$

sending an edge e to a list of pairs of vertices and a (possibly empty) list of (bound) vertices.

Hypergraphs with Binding: Example

Example

Example of a hypergraph with binding consisting of 3 edges, $\{1, 2, 3\}$, one of them having a binding connection, and 5 vertices, $\{a, b, c, d, e\}$:



$$t(1) = \{b\} \quad t(2) = \{d\} \quad t(3) = \{e\}$$
$$s(1) = \{a\} \quad s(2) = \emptyset \quad s(3) = \{c\}$$
$$b(1) = \{\emptyset\} \quad b(2) = \emptyset \quad b(3) = \{\{a\}\}$$

Hypergraphs with Binding: Formal Definition

Definition (Hypergraph with binding)

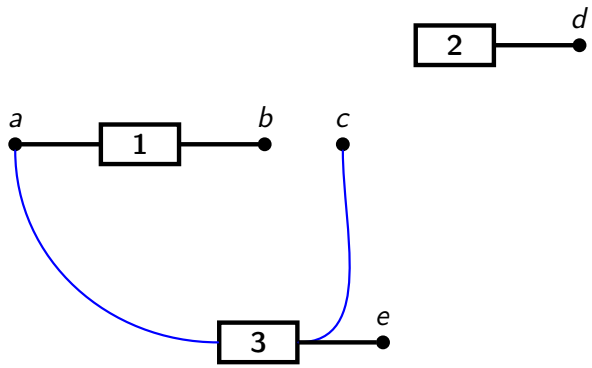
A **hypergraph with binding** is a tuple

$$(E, V, t : E \to V^*, \hat{b} = (s, b) : E \to (V \times V^*)^*),$$

where: E, V are the sets of edges and vertices, respectively, and $t : E \to V^*$, $(s, b) : E \to (V \times V^*)^*$ are the target, source, and binding functions, as described above.

Interfaces

Consider the example again. What are its inputs and outputs?

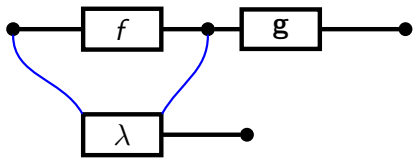


$$\mathit{inputs} = \{c\} \quad \mathit{outputs} = \{b, d, e\}$$

The input interface is then those vertices with in-degree 0, and output interface is those with out-degree 0. Additionally, bound vertices are not part of the input interface — intuitively, bound vertices are in an *inner* layer, so they are not free "inputs" to the hypergraph.

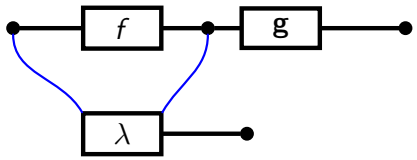
Safety

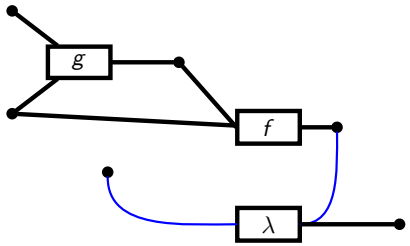
Since there are no restrictions on the connections, we can express all kinds of examples, and some of them might be weird or "unsafe":



Safety

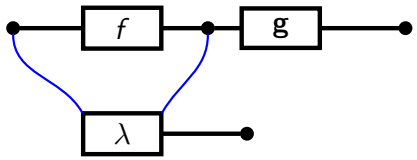
Since there are no restrictions on the connections, we can express all kinds of examples, and some of them might be weird or "unsafe":

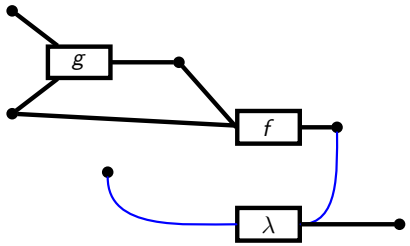


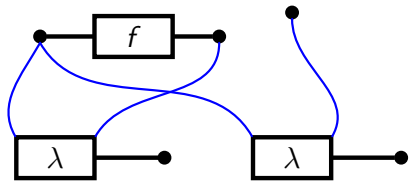


Safety

Since there are no restrictions on the connections, we can express all kinds of examples, and some of them might be weird or "unsafe":



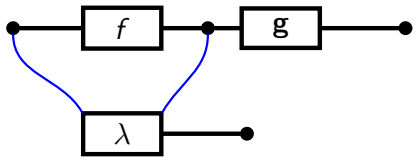


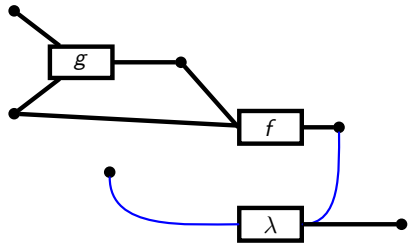


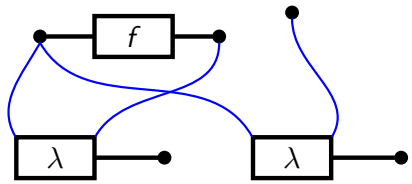
These may correspond to variables being out of scope, or variables being captured by the wrong binder, etc.

Safety

Since there are no restrictions on the connections, we can express all kinds of examples, and some of them might be weird or "unsafe":





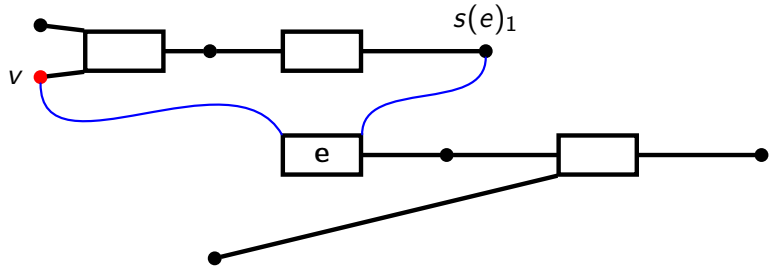


These may correspond to variables being out of scope, or variables being captured by the wrong binder, etc.
In the next slides, we will identify criteria to describe when a hypergraph with binding is well-formed, or "safe", using the notion of *bodies* — "traces" of the bound vertices as they are passed through the graph.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.

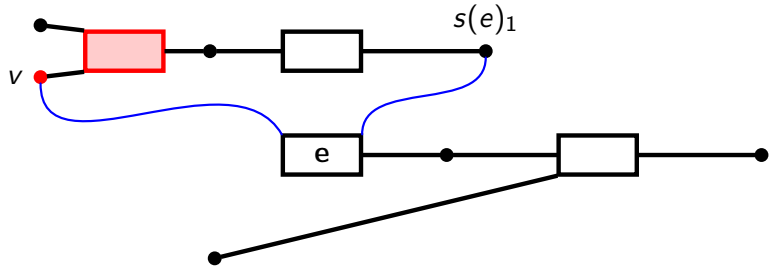


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.
- edges with a source in the body are also part of the body (until we reach the bound source vertex $s_i(e)$): $\forall e', s(e') \cap G_e^i \neq \emptyset$, and $s_i(e) \notin s(e') \cap G_e^i, e' \in G_e^i$.

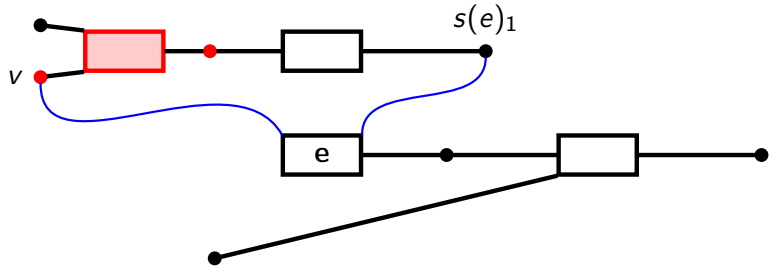


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.
- edges with a source in the body are also part of the body (until we reach the bound source vertex $s_i(e)$): $\forall e', s(e') \cap G_e^i \neq \emptyset$, and $s_i(e) \notin s(e') \cap G_e^i, e' \in G_e^i$.
- targets of edges in the body are also part of the body: $\forall v$ s.t. $\exists e' \in G_e^i : v \in t(e'), v \in G_e^i$.

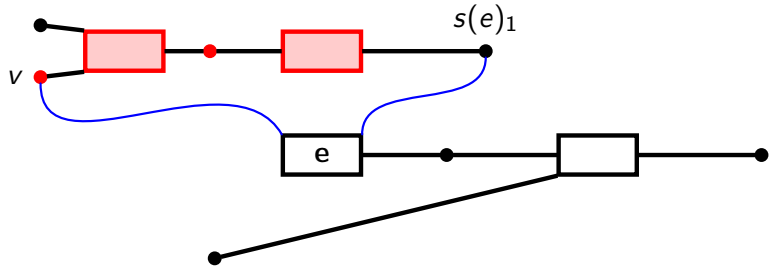


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.
- edges with a source in the body are also part of the body (until we reach the bound source vertex $s_i(e)$): $\forall e', s(e') \cap G_e^i \neq \emptyset$, and $s_i(e) \notin s(e') \cap G_e^i, e' \in G_e^i$.
- targets of edges in the body are also part of the body: $\forall v$ s.t. $\exists e' \in G_e^i : v \in t(e'), v \in G_e^i$.

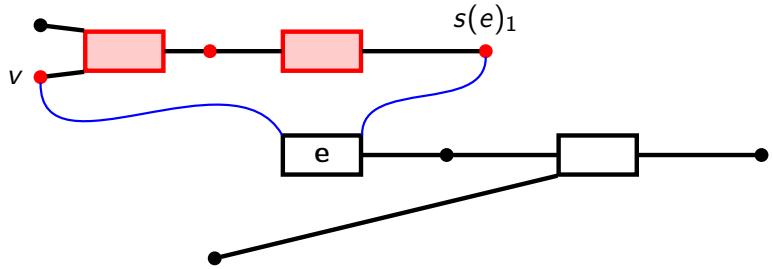


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.
- edges with a source in the body are also part of the body (until we reach the bound source vertex $s_i(e)$): $\forall e', s(e') \cap G_e^i \neq \emptyset$, and $s_i(e) \notin s(e') \cap G_e^i, e' \in G_e^i$.
- targets of edges in the body are also part of the body: $\forall v$ s.t. $\exists e' \in G_e^i : v \in t(e'), v \in G_e^i$.

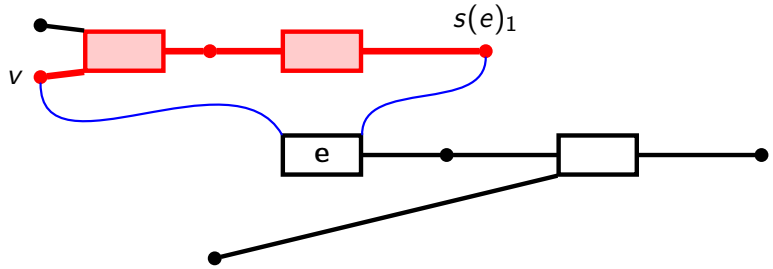


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Bodies

For each edge e and each of its sources $s(e)_i$, we can define the body G_e^i inductively:

- bound vertices are a part of the body: $\forall v \in b_i(e), v \in G_e^i$.
- edges with a source in the body are also part of the body (until we reach the bound source vertex $s_i(e)$): $\forall e', s(e') \cap G_e^i \neq \emptyset$, and $s_i(e) \notin s(e') \cap G_e^i, e' \in G_e^i$.
- targets of edges in the body are also part of the body: $\forall v$ s.t. $\exists e' \in G_e^i : v \in t(e'), v \in G_e^i$.

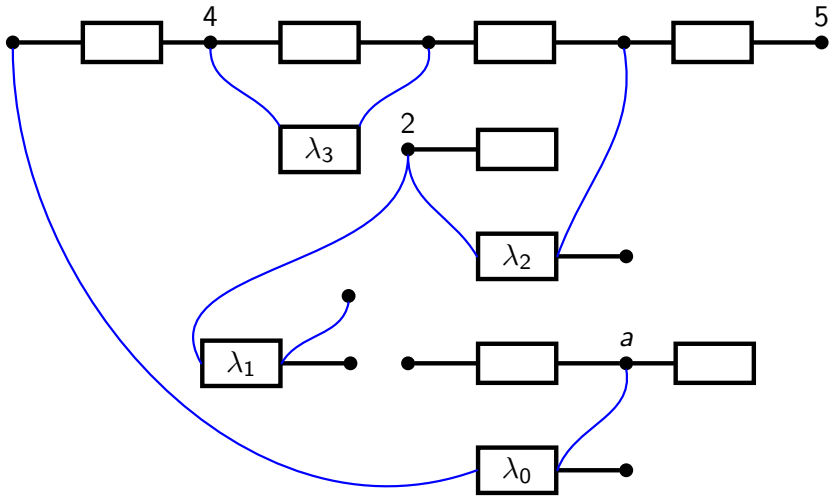


... and go on until reaching the source vertex $s_i(e)$, or the interface.

Example

Example

The hypergraph below is clearly unsafe. The vertices and edges with number labels correspond to conditions for safety not being satisfied. Notice that the vertex a is not a problem - as it is a "free" vertex, it is not part of the body $G^1_{\lambda_0}$.



Safety Criteria

The conditions for a hypergraph with binding to be safe would mean the following:

1. vertices can be bound to only one edge and only once: if $v \in b_i(e)$ for some $e \in E, i \in \mathbb{N}$, and $v \in b_j(e')$ for some $e' \in E, j \in \mathbb{N}$, then $e = e', i = j$.
2. no shared body: $\forall e \neq e',\ \forall i, j : \mathit{Edges}(G_e^i) \cap \mathit{Edges}(G_{e'}^j) = \emptyset$.
3. bound vertices cannot be a target: if $v \in b_i(e)$ for some $e \in E, i \in \mathbb{N}$, then $v \notin t(e')$ for $\forall e' \in E$.
4. no leaking - outputs of the body, connected to the bound vertex, cannot be shared: if $y \in G_e^i$ such that
   $\exists$ *directed path* $[e_0, \ldots, e_1]$ *such that* $s(e_0) \cap b_i(e) \neq \emptyset, y \in t(e_1)$ for some $e \in G, i \in \mathbb{N}$, and $y = s(e)_i$, then $\forall e' \in G,\ y \notin s(e')$.
5. no leaking - outputs of the body, connected to the bound vertex, cannot be in the output interface: if $y \in G_e^i$ such that
   $\exists$ *directed path* $[e_0, \ldots, e_1]$ *such that* $s(e_0) \cap b_i(e) \neq \emptyset, y \in t(e_1)$ for some $e \in G, i \in \mathbb{N}$, then $y \notin O_G$.

Category of Hypergraphs with Binding

Next, we define morphisms, and then the category of hypergraphs with binding, in which we will eventually want to perform rewriting.

Definition

A **morphism** between hypergraphs with binding is a pair of functions $(h, k) : (E_G, V_G) \to (E_H, V_H)$ such that they commute with respect to $t, (s, b)$.

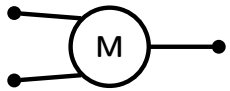
Proposition

*The category **bHyp** of hypergraphs with binding and their morphisms is adhesive.*

This is sufficient for DPO-I rewriting to be well-defined! However, the subcategory of safe hypergraphs with binding is not adhesive. In the full version of the paper, we provide criteria for when DPO-I rewriting in **bHyp** preserves safety.

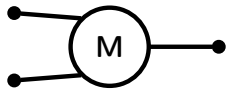
Metavariables

Next, we want to define a substitution operation, and for that, we need to introduce metaedges. They can take their own arguments, so we consider them a subset of edges, labelled from a set of metavariables from the signature.



Metavariables

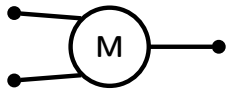
Next, we want to define a substitution operation, and for that, we need to introduce metaedges. They can take their own arguments, so we consider them a subset of edges, labelled from a set of metavariables from the signature.



A substitution operation must take a hypergraph G, a metavariable $M \in G$, a replacement hypergraph T, and yield $G[M/T]$. If M appears n times in G, we must insert n copies of T, sharing the *free* vertices of T across all copies while duplicating its bound internal structure.

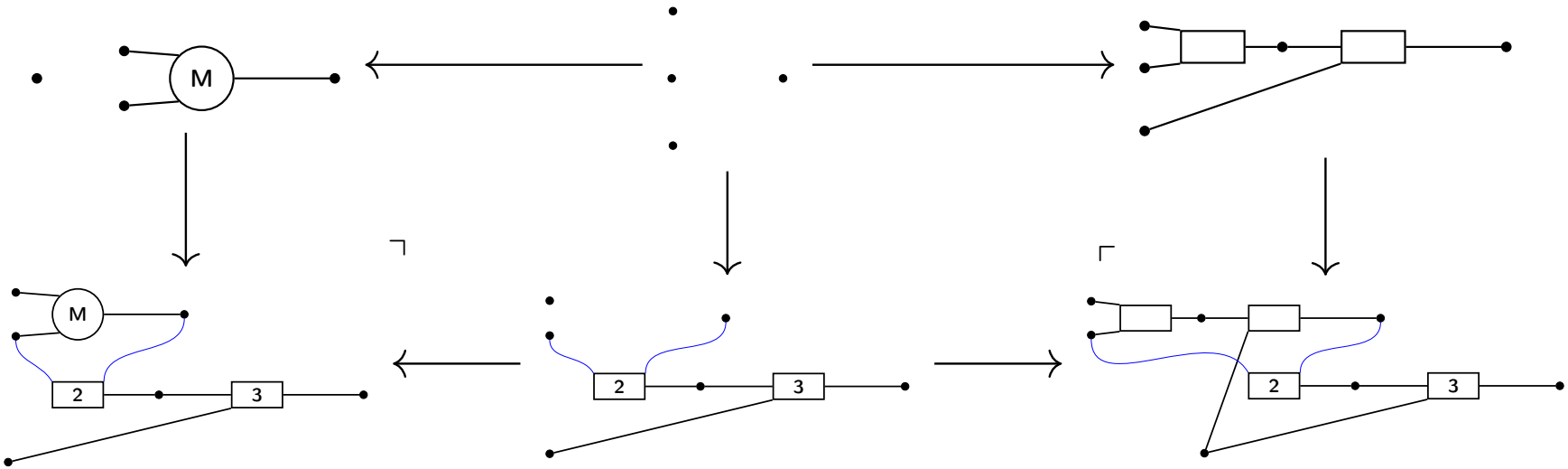
Metavariables

Next, we want to define a substitution operation, and for that, we need to introduce metaedges. They can take their own arguments, so we consider them a subset of edges, labelled from a set of metavariables from the signature.



A substitution operation must take a hypergraph G, a metavariable $M \in G$, a replacement hypergraph T, and yield $G[M/T]$. If M appears n times in G, we must insert n copies of T, sharing the *free* vertices of T across all copies while duplicating its bound internal structure.
In the extended abstract, we provide detailed constructions for extraction of the data needed to define substitution. Here, we only show an example and the definition.

Substitution: Example



Substitution

Definition

Let $I_G \to G \leftarrow O_G$ and $I_T \to T \leftarrow O_T$ be hypergraphs with binding with interfaces, $M \in \mathcal{M}_G$ a metavariable.

Substitution

Definition

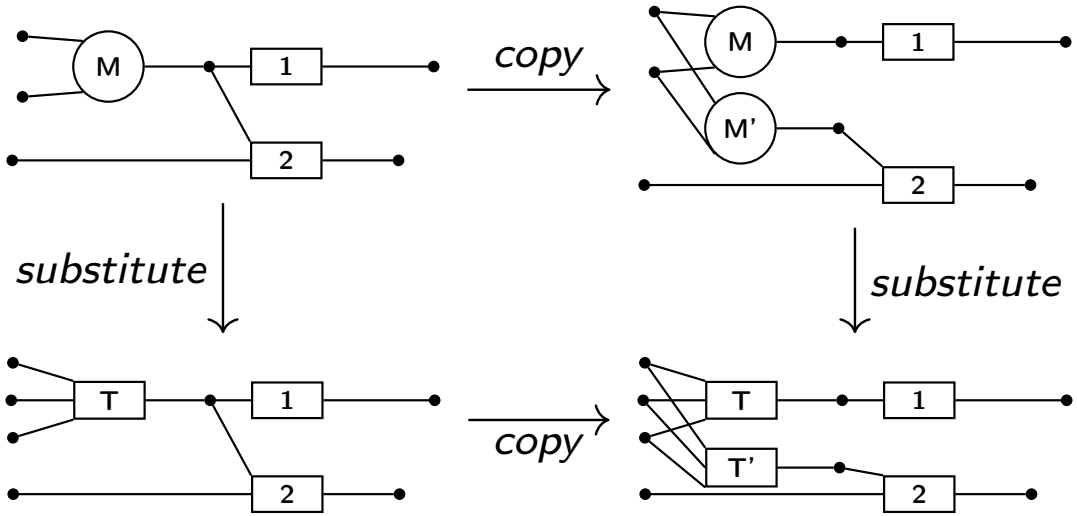
Let $I_G \to G \leftarrow O_G$ and $I_T \to T \leftarrow O_T$ be hypergraphs with binding with interfaces, $M \in \mathcal{M}_G$ a metavariable. A substitution $\theta : M \mapsto T$, $\theta G := I \to G[M/T] \leftarrow O$ is given by: a map $f : I_M \to I_T + O_T \in \mathbf{bHyp}$, and a span $I_G \leftarrow K \to I_{F_T}$, with K a discrete hypergraph of shared variables. Then the substitution is constructed via the following DPO in **bHyp**:

$$\begin{array}{ccccc} n \cdot M \sqcup K & \longleftarrow & n \cdot (I_M) \sqcup K & \longrightarrow & n \hat{\cdot} T \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & C & \longrightarrow & G[M/T] \end{array}$$

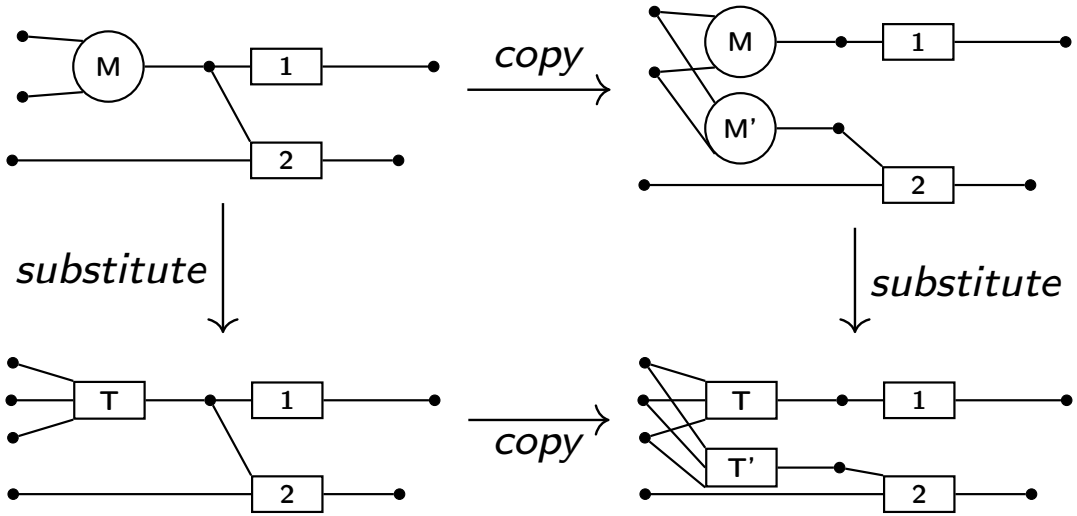
with $I = PO(I_G \leftarrow K \to I_{F_T})$, $O = O_{n \hat{\cdot} T} \sqcup O_G$, $\mathcal{M} = (\mathcal{M}_{\mathcal{G}} \backslash \{M\}) \cup \mathcal{M}_{\mathcal{T}}$.

It is also important to note that we can perform certain operations on hypergraphs (with binding), that change the hypergraph without changing its semantic meaning: namely, copying and glueing of shared sub-graphs.

It is also important to note that we can perform certain operations on hypergraphs (with binding), that change the hypergraph without changing its semantic meaning: namely, copying and glueing of shared sub-graphs.



It is also important to note that we can perform certain operations on hypergraphs (with binding), that change the hypergraph without changing its semantic meaning: namely, copying and glueing of shared sub-graphs.



We say that $G \equiv_{bHyp} H$ when G can be obtained from H via a (finite) sequence of copying and glueing operations.

Context and Rewrite Rules

Now that we have metavariables and substitution, we have a mechanism to represent a family of rewrite rules by a single rule with metavariables. To ensure such rewriting system is well-defined, we need to adapt the notion of context.

The idea is that context is a hypergraph with a "hole", that is a shape, that can be filled by another hypergraph provided the interface matches the hole's shape.

Context and Rewrite Rules

Now that we have metavariables and substitution, we have a mechanism to represent a family of rewrite rules by a single rule with metavariables. To ensure such rewriting system is well-defined, we need to adapt the notion of context.

The idea is that context is a hypergraph with a "hole", that is a shape, that can be filled by another hypergraph provided the interface matches the hole's shape. We define it as an equivalence class of pushout squares, with the pushout complement and the "hole" interface fixed:

$$\begin{array}{ccc} I_1 & \longrightarrow & C_1 \\ \downarrow & & \downarrow \\ H_1 & \xrightarrow{f_1} & G_1 \end{array}$$

where $I_1 \to H_1$ is an interface map and C_1 is the pushout complement representing G_1 with a H_1-shaped hole. We say that $f_1 \sim f_2$ when the top rows of their squares are the same modulo $=_{bHyp}$.

Rewriting: Example

To apply a rewrite rule then, we need the following data:

- a rewrite rule with metavariables,
- a substitution θ for the metavariables in the rule to instantiate it,
- a matching morphism from the instantiated left-hand side L to the hypergraph we want to rewrite, such that it is equivalent to a context that allows such rewriting.

Rewriting: Example

To apply a rewrite rule then, we need the following data:

- a rewrite rule with metavariables,
- a substitution θ for the metavariables in the rule to instantiate it,
- a matching morphism from the instantiated left-hand side L to the hypergraph we want to rewrite, such that it is equivalent to a context that allows such rewriting.

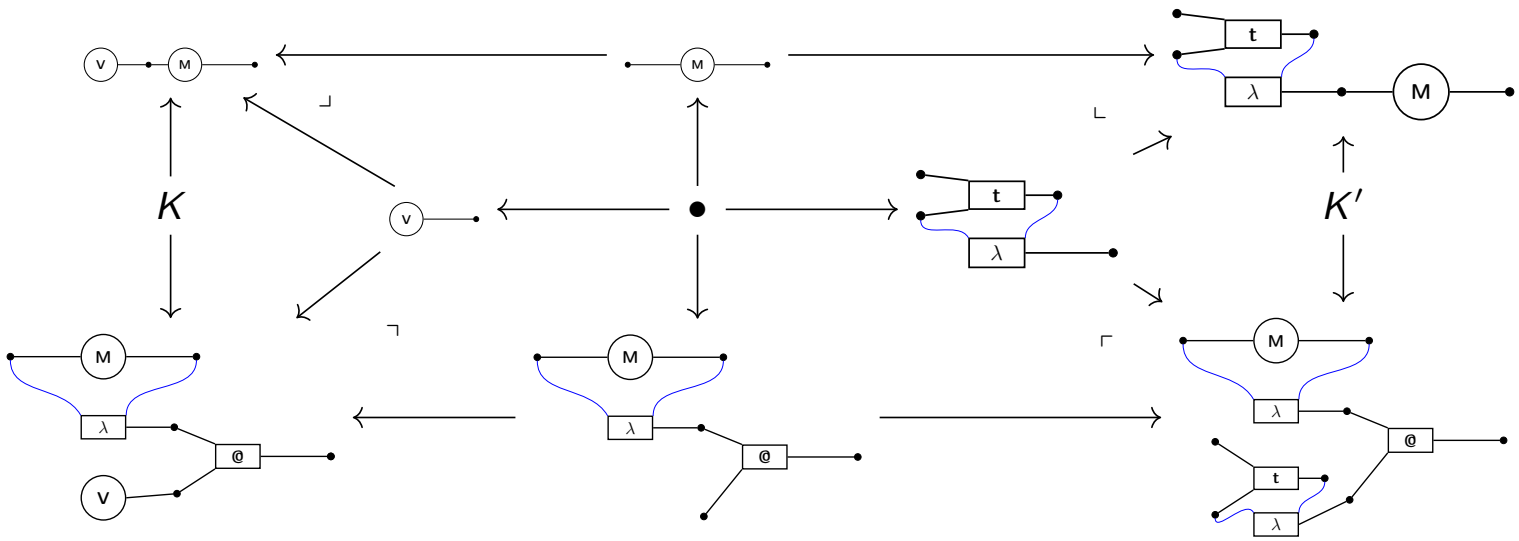
Going back to the starting example: $(\lambda x.M[x])V \to M[V]$, with a substitution $V \mapsto \lambda x.t$ for some t.

Rewriting: Example

To apply a rewrite rule then, we need the following data:

- a rewrite rule with metavariables,
- a substitution θ for the metavariables in the rule to instantiate it,
- a matching morphism from the instantiated left-hand side L to the hypergraph we want to rewrite, such that it is equivalent to a context that allows such rewriting.

Going back to the starting example: $(\lambda x.M[x])V \to M[V]$, with a substitution $V \mapsto \lambda x.t$ for some t.



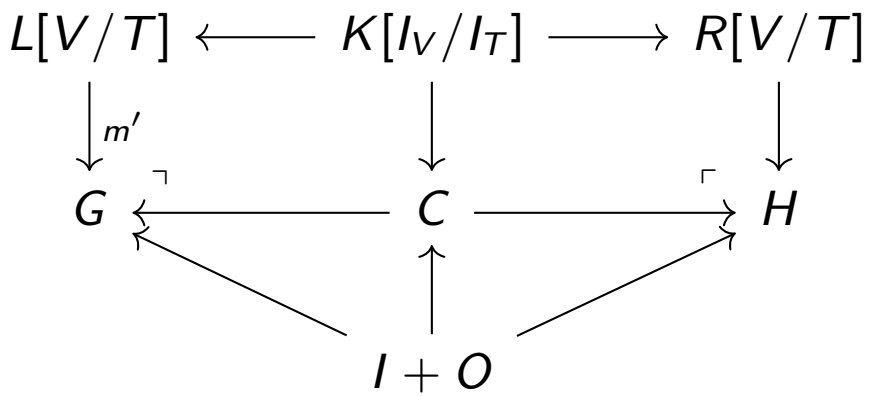
Rewriting: Definition

Definition (Rewriting in context)

Let $Ectx$ be a set of contexts, $L \leftarrow K \rightarrow R$ a rewrite rule with L, R having the same metaedges and the same interface. An **application of the rewrite rule** $L \leftarrow K \rightarrow R$ to a hypergraph with binding G is given by:

- substitutions $V_i \mapsto T_i$ for metaedges $V_i \in M_L$;
- a matching map $L[V/T] \xrightarrow{m} G$ such that $[m] \in Ectx$.

The result of the rewrite is then constructed via the usual DPO-I diagram for $L[V/T] \leftarrow K \rightarrow R[V/T]$.



Second-Order Computational Systems

A **second-order CS** ([3, Section 3]) is a tuple $(\Sigma, \mathcal{M}, \mathcal{R})$, consisting of a binding signature Σ of function symbols, a set of metavariables $\mathcal{M}$, and a set $\mathcal{R}$ of rewrite rules.

Second-Order Computational Systems

A **second-order CS** ([3, Section 3]) is a tuple $(\Sigma, \mathcal{M}, \mathcal{R})$, consisting of a binding signature Σ of function symbols, a set of metavariables $\mathcal{M}$, and a set $\mathcal{R}$ of rewrite rules. The raw meta-terms are given by the grammar $t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid M[t_1, \ldots, t_n]$, while well-formed meta-terms are those derived by the rules that ensure arities and variable scopes are respected.

$$\frac{1 \leq i \leq n}{Z \triangleright n \vdash i} \qquad \frac{M \in Z(m) \quad Z \triangleright n \vdash t_i \quad (1 \leq i \leq m)}{Z \triangleright n \vdash M[t_1, \ldots, t_m]}$$

$$\frac{\begin{array}{c} f : \langle i_1, \ldots, i_m \rangle \in \Sigma \\ Z \triangleright n + i_j \vdash t_j \quad (1 \leq j \leq m) \end{array}}{Z \triangleright n \vdash f(\, n{+}1 \cdots n{+}i_1.t_1,\ \ldots,\ n{+}1 \cdots n{+}i_m.t_m\,)}$$

Typing rules of meta-terms, [3, Figure 1]

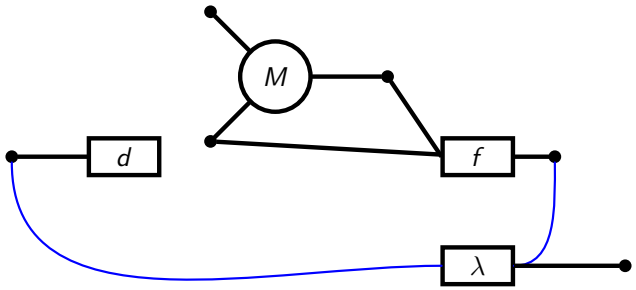
$$(\text{Rule})\ \frac{\begin{array}{c} \triangleright n + i_j \vdash s_j \quad (1 \leq j \leq k) \quad \theta = [M_1 \mapsto s_1, \ldots, M_m \mapsto s_k] \\ (M_1^{(i_1)}, \ldots, M_k^{(i_k)}) \triangleright 0 \vdash \ell \Rightarrow r) \in \mathcal{C} \end{array}}{\triangleright n \vdash \theta^\sharp(\ell) \to_{\mathcal{C}} \theta^\sharp(r)}$$

$$(\text{Fun})\ \frac{f : \langle i_1, \ldots, i_m \rangle \in \Sigma \quad \triangleright n + i_j \vdash t_j \to_{\mathcal{C}} t'_j \quad (\text{some single } j \text{ s.t. } 1 \leq j \leq k)}{\triangleright n \vdash f(\ldots, n{+}1 \cdots n{+}i_j.t_j, \ldots) \to_{\mathcal{C}} f(\ldots, n{+}1 \cdots n{+}i_j.t'_j, \ldots)}$$

Rewrite rule application, [3, Figure 2]

Example

The translation $(-)^\dagger$ from SOCS meta-terms into $\mathbf{bHyp}_{\Sigma \cup \{d\}}$ is intuitive from the definition. The additional label 'd' is necessary to prevent the translation of variables from appearing in the interface.
For example, a translation of a meta-term $\lambda(x.f(M[z, y], y))$ is:



Translation Theorem

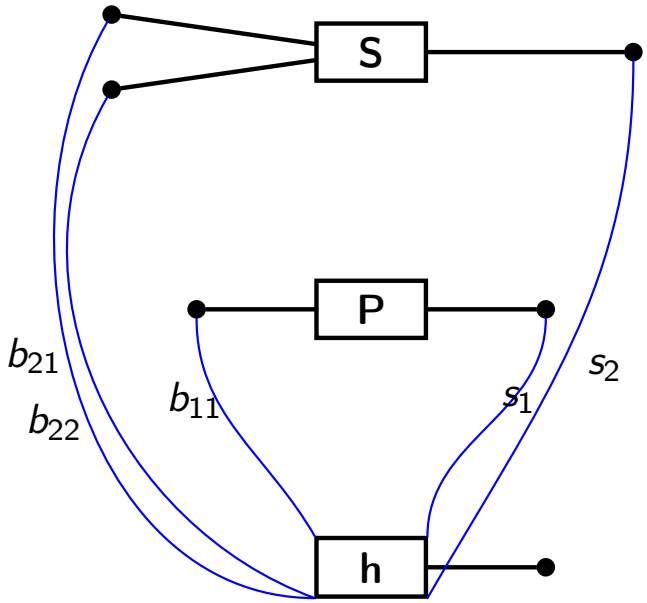
Theorem

Let $(\Sigma, \mathcal{M}, \mathcal{R})$ be a second-order CS. For each meta-term t, there exists a hypergraph with binding $t^\dagger$ with the signature $\Sigma \cup \{d\}$. Moreover, the following holds.

1. *(Safety). If t is well-formed, $t^\dagger$ is safe.*
2. *(One-step rewrite). Each one-step rewrite $\rhd\, n \vdash s \to_C t$ translates to a rewrite rule application in $\mathbf{bHyp}_{\Sigma \cup \{d\}}$ via $(-)^\dagger$.*
3. *(Inverse one-step rewrite). For each rewrite rule R in $\mathbf{bHyp}$ $L \leftarrow K \to R$, $\theta : L \ni V \mapsto T$, $m : L \hookrightarrow G$, there is a rewrite rule application in the second-order CS $(\Sigma, \mathcal{M}, \mathcal{R})$ such that it translates to R via $(-)^\dagger$ modulo $=_{bHyp}$.*

Another Example

Example with a function symbol having multiple arguments with binding, from [5]: Let $h :< 1, 2 >$ be a function symbol representing a handler. Then the term $h(x.P, y.k.S)$ would be represented as a hypergraph with binding like that:



Rewriting Morphisms in a Monoidal Closed Category

Another example relates our work to string diagram rewriting. Hypernets, as special hierarchical hypergraphs, are introduced as a suitable formalisation of string diagrams for monoidal closed categories (MCC in short) in [1], and also as a model for λ-abstraction.
We relate hypernets, which do not form an adhesive category, to hypergraphs with binding, and show how to simulate MCC in **bHyp**.

Rewriting Morphisms in a Monoidal Closed Category

Another example relates our work to string diagram rewriting. Hypernets, as special hierarchical hypergraphs, are introduced as a suitable formalisation of string diagrams for monoidal closed categories (MCC in short) in [1], and also as a model for λ-abstraction.
We relate hypernets, which do not form an adhesive category, to hypergraphs with binding, and show how to simulate MCC in **bHyp**.
A **monoidal (right) closed category** is a monoidal category $\mathcal{C}$ satisfying: **1,** for every pair of objects B, C there is an object $B \Rightarrow C$ and a morphism $ev_{B,C} : (B \Rightarrow C) \otimes B \to C$; and **2,** for every triple of objects A, B, C there is an operation $\Lambda_{A,B,C} : \mathcal{C}(A \otimes B, C) \to \mathcal{C}(A, B \Rightarrow C)$ satisfying three equations for all $f : A \otimes B \to C, g : Z \to A$:

- $f = \Lambda_{A,B,C}(f) \otimes 1_B ; ev_{B,C}$;
- $1_{B \Rightarrow C} = \Lambda_{B \Rightarrow C,B,C}(ev_{B,C})$;
- $\Lambda_{Z,B,C}(g \otimes 1_B ; f) = g ; \Lambda_{A,B,C}(f)$.

The rules of MCC are represented by rewrite rules on hypernets.

Hypernet definitions

Definition (Hierarchical Hypergraph, [1])

A **hierarchical hypergraph** is a tuple $(V, E, s, t, l_V, l_E, p_V, p_E)$ where V, E, s, t are as in a hypergraph, the edge labelling is modified to include an extra value $l_E : E \to \Sigma_E + 1$, the vertex labelling allows complex labels such as $A \Rightarrow B, A \otimes B$ with operations $\bullet \Rightarrow \bullet, \otimes$ being defined on the set of vertex labels, and parental functions $p_V : V \to E + 1$, $p_E : E \to E + 1$. The edge and vertex labelling must align with vertex labels being the input and output types of edge labels. The parental functions satisfy the following conditions:

- $p_V(v) = p_E(e) = p_V(v')$ for all $v \in s(e)$, $e' \in t(e)$.
- the parent relation is acyclic: for all $e \in E$, there is some $k \geq 1$ such that $(p_{E,\perp})^k(e) = \perp$ where $\perp$ is the element of 1 and $p_{E,\perp} : E + 1 \to E + 1$ is the extension of p_E by adding $p_{E,\perp}(\perp) = \perp$.

There will be examples on the next slide. The idea is that now edges can be "labelled" by other hypergraphs, as well as functions from signature.

Rewriting Morphisms in a Monoidal Closed Category

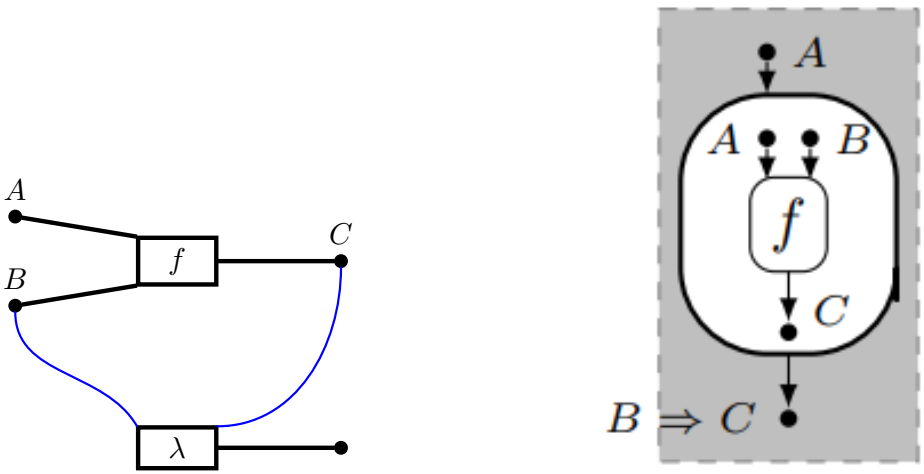
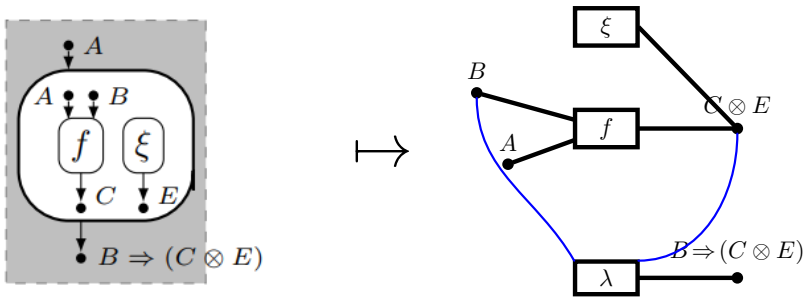


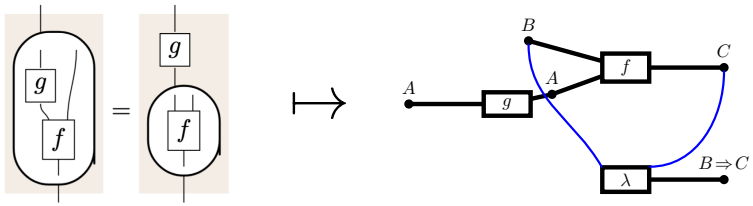
Figure: $\Lambda(f)$ for **bHyp** and string diagrams (Figure from [1])

Example

We define a translation $(\cdot)^\dagger$ mapping a hierarchical hypergraph into $\mathbf{bHyp}_{\Sigma_E \cup \{\lambda\}}$, where $\lambda : \langle 1 \mid 1 \rangle$ acts as the boundary of the abstraction:



(a) Translation of a hypernet.



(b) Translation of the slide rule.

Figure: Pictures of hypernets are taken from [1, Figure 9(c), Figure 3(c)].

Notice that, similarly to composition or product of morphisms, the slide rule doesn't need a special rewrite rule to hold in **bHyp**, as it is already captured by construction, emphasising that it is an additional structure on a category, rather than behaviour of a special morphism.

Translation Theorem

Theorem

For each hierarchical hypergraph $H = (E_H, V_H, s, t, p, l_E : E \to \Sigma_E + 1)$, there exists a hypergraph with binding $H^\dagger = (E_{H^\dagger}, V_{H^\dagger}, s, t, b)$, with the signature $\Sigma_E \cup \{\lambda\}$ where $\lambda : \langle 1|1 \rangle$. Moreover, the following holds.

1. *(Safety). If H is a hypernet, $H^\dagger$ is safe.*
2. *(β, η laws). For each hypernet rewrite rule $L \leftarrow K \to R$ that is either a β law or an η law, it translates to a rewrite rule of hypergraphs with binding via $(-)^\dagger$.*
3. *(Inverse rewrite). For any rewrite rule $L \leftarrow K \to R$ of safe hypergraphs with binding, a matching morphism $m : L \hookrightarrow G$ such that the context $\mathcal{C} \ni m$ is convex, there exists a hypernet rewrite rule $\mathcal{L}, \mathcal{R}$ such that $\mathcal{L}^\dagger =_{bHyp} L, \mathcal{R}^\dagger =_{bHyp} R$. Moreover, for the result of the rewrite rule application at the convex matching $i : \mathcal{L} \hookrightarrow \mathcal{G}$, it holds that $\mathcal{G}[\mathcal{L} \mapsto \mathcal{R}]^\dagger =_{bHyp} G[L \mapsto R]$. This construction is unique up to the slide rule application and λ-clauses for vertex labels $A \Rightarrow B$.*

Future Work

Our primary future direction is to develop a proof methodology of improvement, as described in the motivation, based on the rewriting systems of hypergraphs with binding. We expect that we could transfer the automatable proof methodology for second-order term rewriting [5, 4], to rewriting of hypergraphs with binding, potentially relying on critical pair analysis.

	[2]	[5, 4]	this work
combinatorial representation of programs	✓	×	✓
solid formalisation of rewriting	×	✓	✓
proof methodology of improvement	✓	✓	×
—robustness of proof methodology	✓	×	–
—automation of proof methodology	×	✓	–

Thank you!


Mario Alvarez-Picallo, Dan Ghica, David Sprunger, and Fabio Zanasi.
Rewriting for monoidal closed categories.
7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022), 2022.


Dan R. Ghica, Koko Muroya, and Todd Waugh Ambridge.
A robust graph-based approach to observational equivalence.
Log. Methods Comput. Sci., 21(2), 2025.


Makoto Hamana.
Complete algebraic semantics for second-order rewriting systems based on abstract syntax with variable binding.
Mathematical Structures in Computer Science, 2022.


Makoto Hamana and Kento Emoto.
Recheck: Automated contextual improvement verifier for functional calculi across user-defined operational semantics.
In Sebastian Junges and Guy Katz, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 32nd International*

Conference, TACAS 2026, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2026, Turin, Italy, April 11-16, 2026, Proceedings, Part II, Lecture Notes in Computer Science, pages 215–234. Springer, 2026.

Koko Muroya and Makoto Hamana.
Term evaluation systems with refinements: First-order, second-order, and contextual improvement.
In Jeremy Gibbons and Dale Miller, editors, *Functional and Logic Programming - 17th International Symposium, FLOPS 2024, Kumamoto, Japan, May 15-17, 2024, Proceedings*, volume 14659 of *Lecture Notes in Computer Science*, pages 31–61. Springer, 2024.