A bigraphical model of the simply typed λ-calculus

Shane Ó Conchúir

Foundations and Methods Group
School of Computer Science and Statistics
Trinity College Dublin, Ireland
Shane.OConchuir@cs.tcd.ie

Abstract

We present a model of the simply typed λ-calculus as a simply typed explicit substitution calculus encoded in a bigraphical reactive system. The reactive system combines a previous model by Milner with a sorting of the place graph structure which is defined using a generalisation of our previous work on kind sortings. The model demonstrates the expressiveness of these sortings.

We identify some useful subcategories of kind sorted bigraphs which retain some important properties of the transition theory of pure bigraphs. We introduce a more general notion of kind sorting which can properly model XML data. We also present a simple idea of combining sortings which retains the transition theory and consider combining kind sorting with the Plato-graphical sorting of Birkedal et al.

1 Introduction

This paper presents a model of the simply typed λ-calculus as a simply typed explicit substitution calculus encoded in a bigraphical reactive system (Brs). We first introduce simple types into Milner's Λ_{sub} calculus using the typing rules for λxgc [5] presented by Di Cosmo and Kesner [13]. We then advance the definitions of kind sortings [9] to isolate sub-s-categories of sorted bigraphs which can be used to model both finite CCS and simply typed Λ_{sub}. Finally, we present our encoding of simply typed Λ_{sub} in bigraphs. Our motivation is simply to present an example that demonstrates that kind sorting is quite expressive. We believe that a model of the simply typed λ-calculus demonstrates this.

The central theme of this paper is the model of simply typed λ-calculus. However, the appendices contain further generalisations of kind sortings and make connections to related work which we believe could be useful in the future.

To keep this paper brief, we assume a familiarity with pure bigraphs [26]. We will make mention of local bigraphs [27] but do not need to go into details. We restrict the discussion to hard bigraphs, bigraphs where every non-atomic node has a child. This is sensible in a model of the λ-calculus. We also assume knowledge of the λ-calculus and explicit substitutions. We refer the reader to Barendregt's book [1] and Rose's tutorial [29] for these subjects respectively. For more recent work on explicit substitution, we refer the reader to Kesner's paper [18] which describes the progress made in the interim and discusses the relationship between explicit substitution calculi and Linear Logic's proof-nets.

1.1 Structure of the paper

We begin in Section 2 by introducing the simply typed version of Milner's Λ_{sub} calculus, denoted $\Lambda^{\tau}_{\text{sub}}$. Λ_{sub} was previously modelled by Milner as a bigraphical reaction system $'\Lambda\text{BIG}$.

Milner's definition of place sorting [26] and a slight generalisation of kind sorting are given in Section 3. Our new definition of kind sorting introduces a notion of *invisible controls*, controls which are forced to be hidden inside the place graph structure and never allowed to be children of a root. This paves the way to the definition of Milner's multi-nodes as elements of kind signatures. As we will use a subcategory of kind sorted bigraphs to model $\Lambda^{\tau}_{\text{sub}}$, we introduce general definitions for subcategories of kind sorted bigraphs. We state whether or not the various subcategories have the properties of RPO creation and pushout reflection used in the transition theory of bigraphs.

In Section 4, we describe the kind signature for the bigraphical model $'\Lambda\text{BIG}^{\tau}$ of $\Lambda^{\tau}_{\text{sub}}$. The signature is based on the typing rules for Λ_{sub} and uses an infinite number of 'typed' $'\Lambda\text{BIG}$ controls. We describe our motivation for using a subcategory of kind sorted bigraphs and choose a subcategory which is similar to (but generalises) the type of subcategory Milner used to model finite CCS. This later provides a nice correspondence between the type of a Λ_{sub} term under some environment Γ and the interface sort of its encoding in $\Lambda^{\tau}_{\text{sub}}$.

The encoding of $\Lambda^{\tau}_{\text{sub}}$ into $'\Lambda\text{BIG}^{\tau}$ is presented in Section 5. We introduce a partial functor from $'\Lambda\text{BIG}^{\tau}$ to $'\Lambda\text{BIG}$ which recovers the bigraphical encoding of untyped Λ_{sub} terms given by Milner. We finally reason about normalisation and confluence properties for simply typed Λ_{sub} based on the untyped calculus. This allows us to proof, using a technique due to Herbelin [15], that the simply typed calculus is strongly normalising. This result adds to the positive properties of Λ_{sub} which include simulation of β-reduction, closed confluence, preservation of strong normalisation, and full composition of substitution [10]. We use these results to reason about encodings of simply typed terms in $'\Lambda\text{BIG}^{\tau}$.

The central theme to this paper is the presentation of $'\Lambda\text{BIG}^{\tau}$ but its investigation has spun off some other questions and ideas which we present in the appendices. This is also where we intern (for the sake of the reader) some of our intuitions.

As we use a subcategory of kind bigraphs for $'\Lambda\text{BIG}^{\tau}$, we had to consider whether subcategories of kind sorted bigraphs retain RPO creation or gain pushout reflection – the full category of kind sorted bigraphs creates RPOs but does not reflect pushouts. Having identified various subcategories in Section 3.3, we prove/disprove these properties in Appendix A.

$'\Lambda\text{BIG}^{\tau}$ can model $\Lambda^{\tau}_{\text{sub}}$ but not all bigraphs in the s-category correspond to a a simply typed term. The grammar of Λ_{sub} is quantified – one subterm lies below an abstraction and two lie below an application or explicit substitution. Pure and kind bigraphs cannot express capacities of controls *i.e.* that a node of a specific control can contain a certain number of controls of some type. Appendix B.1 introduces a generalisation of kind sorting which allows minimum and maximum capacities to be expressed with some degree of freedom. One application of this sorting is modelling XML data whose document order is relevant.

Many sortings are currently being investigated for bigraphs for different purposes. In Appendix B.2, we introduce a simple way of combining sortings. We then present a pairing of kind sorting with the rigid control-sorting of Birkedal et al. [3]. The pairing creates RPOs. In Appendix B.3, we consider using this pairing in the location-aware printing system example of Birkedal et al. to remove some bigraphs from their system which do not reflect the intended model. We also give a simple example of how the expressivity of kind sorted reaction rules could be used in their system. In the same work, Birkedal et al. presented a bigraphical tree-traversal 'algorithm.' In Appendix B.4, we use their idea to present a novel algorithm which utilises the expressivity of kind sorting.

We implicitly use a link sorting in our model of $'\Lambda\text{BIG}^{\tau}$. Appendix C formally presents this link sorting and proves RPO creation.

Appendix D compares Milner's homomorphic sorting with kind sorting in order to provide a better explanation of the latter.

Appendix E presents an idea of building typed $'\Lambda\text{BIG}$ terms with typing rules.

2 Simply typed Λ_{sub}

The set Λx of terms of the untyped Λ_{sub} is defined inductively by

$$t ::= x \quad | \quad \lambda x.t \quad | \quad tt \quad | \quad t[x/t]$$

where the notation $[x/t]$ represents an explicit substitution. Abstractions and explicit substitutions bind their variable *e.g.* x, and we assume a variable convention where the bound variables of a term are distinct and different from the free variables, following Barendregt [1].

Definition (Reduction rules for Λ_{sub})**.** *The reduction relation* $\longrightarrow_{\text{ACD}}$ *of* Λ_{sub} *is defined as the contextual closure modulo* $\equiv$ *of the union of the following reductions.*

$(\lambda x : A.t)u$	$\longrightarrow_{\text{A}}$	$t[x/u]$		*(substitution generation)*
$C[x][x/u]$	$\longrightarrow_{\text{C}}$	$C[u][x/u]$	*this x is free in $C[x]$*	*(non-local substitution)*
$t[x/u]$	$\longrightarrow_{\text{D}}$	t	*if $x \notin \mathsf{FV}(t)$*	*(garbage collection)*

The type system for simply typed Λ_{sub} ($\Lambda^{\tau}_{\text{sub}}$) is as follows. Given an arbitrary non-empty set of ground types $\mathcal{G}$, the set of types is given by the inductive definition

$$\tau := \mathcal{G} \mid \tau \to \tau.$$

We use G and G' to denote ground types, α to denote function types, and A, B, and C to denote arbitrary types. We use the same typing rules that Di Cosmo and Kesner introduced for λxgc [13] which extend the usual simple typing by adding a rule for explicit substitutions. Just as untyped Λ_{sub} has the same set of terms as untyped λxgc, their simply typed variants share the same set of terms. α-equivalence is defined as usual, preserving types.

We consider type environments Γ as functions from a set of variables to the set of types τ.

Lemma 1 (Subject reduction)**.** *If* $\Gamma \vdash t : A$ *and* $t \longrightarrow_{\text{ACD}} u$ *then* $\Gamma \vdash u : A$.

Proof. The proof for $\longrightarrow_{\text{A}}$ and $\longrightarrow_{\text{D}}$ is straightforward. The $\longrightarrow_{\text{C}}$ case may be broken down into six cases depending on whether C is empty or whether x lies directly beneath an abstraction, application, or explicit substitution in the abstract syntax tree. These cases are straightforward, replacing $\Gamma \vdash x : A$ with the derivation of $\Gamma \vdash u : A$ in the proof tree. □

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\texttt{axiom}) \qquad \frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A}\ (\texttt{subs})$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ (\texttt{app}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash x.t : A \to B}\ (\texttt{abs})$$

Figure 1: Typing rules for Λ_{sub}

3 Place sorting

We start this section on sorting by repeating Milner's definition of place-sorting [26]. We then introduce the definition of kind sortings, a particular type of place-sorting. Subcategories of kind sorted bigraphs are shown to retain properties of pure bigraphs which have positive implications for the reaction semantics of reactive systems over these subcategories.

Notation. *Although incorrect, we write subcategory instead of sub-s-category for convenience. If $\mathcal{I}$ is an inclusion functor then we typically abuse notation and write $\mathcal{U}$ for any composite functor $\mathcal{U} \circ \mathcal{I}$. We let $\mathcal{K}$ denote both a bigraphical signature and the set of controls of that signature.*

3.1 Place-sortings

A place-sorting is a sorting discipline which constrains the parent map of a bigraph, admitting only those bigraphs which satisfy the rules of the discipline. Place-sortings must satisfy certain conditions in order that they can be used to form an s-category. Some useful place-sortings have further properties such as that the RPO construction for place-sorted bigraphs should also be based on the RPO construction for pure bigraphs.

In the following, Θ denotes a non-empty set of *sorts*, and θ ranges over Θ.

Definition 1 (place-sorted bigraphs [26])**.** *An interface with width m is* Θ-(place-)sorted *if it is enriched by ascribing a sort to each place $i \in m$. If I is place-sorted, its underlying unsorted interface is denoted by $\mathcal{U}(I)$.*

$\text{´}\textsc{Big}_{\text{h}}(\mathcal{K}, \Theta)$ denotes the s-category over signature $\mathcal{K}$ in which the objects are place-sorted interfaces, and each arrow $G : I \to J$ is a bigraph $G : \mathcal{U}(I) \to \mathcal{U}(J)$. The identities, and composition and tensor product are as in $\text{´}\textsc{Big}_{\text{h}}(\mathcal{K})$, but with sorted interfaces.

We denote place-sorted interfaces as $\langle m, \vec{\theta}, X \rangle$ where $\vec{\theta}$ is a vector $\{\theta_0, \ldots, \theta_{m-1}\}$ ascribing a sort θ_i to each place $i \in m$ of the interface.

Definition 2 (place-sorting [26])**.** *A* place-sorting *is a triple*

$$\Sigma = (\mathcal{K}, \Theta, \Phi)$$

where Φ is a condition on Θ-sorted bigraphs over $\mathcal{K}$. The condition Φ must be satisfied by the identities and preserved by composition and tensor product.

A bigraph in $\text{´}\textsc{Big}_{\text{h}}(\mathcal{K}, \Theta)$ is Σ-(place-)sorted *if it satisfies Φ. The Σ-sorted bigraphs form a subcategory of $\text{´}\textsc{Big}_{\text{h}}(\mathcal{K}, \Theta)$ denoted by $\text{´}\textsc{Big}_{\text{h}}(\Sigma)$. Further, if $\text{´}\mathcal{R}$ is a set of Σ-sorted reaction rules then $\text{´}\textsc{Big}_{\text{h}}(\Sigma, \text{´}\mathcal{R})$ is a* Σ-sorted *Brs.*

Associated with a place-sorting is a forgetful functor

$$\mathcal{U} : \text{´}\textsc{Big}_{\text{h}}(\Sigma) \to \text{´}\textsc{Big}_{\text{h}}(\mathcal{K})$$

which discards sorts and whose codomain is an s-category of pure bigraphs. Such a functor $\mathcal{U}$ is called a *sorting* functor, is surjective on objects, and is faithful. This functor can be used to prove properties of the sorting based on the pure bigraphs. The following two properties are desirable for sorting functors.

Definition 3 (creating RPOs, reflecting pushouts)**.** *Let $\mathcal{F}$ be any functor on an s-category $\text{´}\mathbf{A}$. Then $\mathcal{F}$* creates RPOs *if, whenever $\vec{D}$ bounds $\vec{A}$ in $\text{´}\mathbf{A}$, then any RPO for $\mathcal{F}(\vec{A})$ relative to $\mathcal{F}(\vec{D})$ has a unique $\mathcal{F}$-preimage that is an RPO for $\vec{A}$ relative to $\vec{D}$.*

$\mathcal{F}$ reflect pushouts *if, whenever $\vec{D}$ bounds $\vec{A}$ in $\text{´}\mathbf{A}$ and $\mathcal{F}(\vec{D})$ is a pushout for $\mathcal{F}(\vec{A})$, then $\vec{D}$ is a pushout for $\vec{A}$.*

Work by Leifer and Milner [21] and Ole Høgh Jensen led to the following.

Theorem 2 (useful place-sortings [25]). *In* $\acute{\mathcal{B}}\text{IG}_{\text{h}}(\Sigma, \acute{\mathcal{R}})$*:*

1. *If* Σ *creates RPOs then* $\sim_{\text{ST}}$ *is a congruence.*
2. *If in addition* Σ *reflects pushouts and* $\acute{\mathcal{R}}$ *is simple prime affine, then* PE *is adequate for* ST*.*

The definitions of $\sim_{\text{ST}}$, ST, and PE are beyond this document but the gist of this result is that when the antecedents hold, the theorem can be used to show that: 1) bisimilarity is a congruence and; 2) that one may 'reduce' some of the labelled transition systems without affecting bisimilarity. This is why RPO creation and pushout reflection are desirable.

3.2 Kind sorting

Kind sorting [9] generalises the notion of atomicity in the place graph structure (the hierarchical tree-like structure) of bigraphs. The controls in a pure bigraphical system are either non-atomic (can contain nodes of any control) or atomic (cannot contain anything). The controls of a kind system – a system under a kind sorting – may contain nodes of some *subset* of the set of controls. Although we have defined and explored kind sortings, the initial idea was proposed by Jensen and Milner [17].

Definition 4 (kind signature). *A* kind signature $\{\mathcal{K}, arity, actv, vsbl, kind\}$ *is composed of a set* $\mathcal{K}$ *of* controls *and four maps:*

$$\begin{aligned} arity &: \mathcal{K} \to \mathbb{N} \\ actv &: \mathcal{K} \to \{\mathsf{passive}, \mathsf{active}\} \\ vsbl &: \mathcal{K} \to \{\mathsf{vis}, \mathsf{inv}\} \\ kind &: \mathcal{K} \to \mathcal{P}(\mathcal{K}). \end{aligned}$$

arity is the usual function assigning a number of ports to a control. We call $kind(K) = sort(K)$ the *kind* or *sort* of K. It is the set of controls that K-node can contain in a sorted bigraph. A control K is said to be *atomic* if its sort is $\emptyset$, otherwise *non-atomic*. Atomic controls may not be active. The function *vsbl* partitions $\mathcal{K}$ into two sets $\mathcal{K}_{\mathsf{vis}}$ and $\mathcal{K}_{\mathsf{inv}}$ of *visible* and *invisible* controls.

Definition 5 (kind sorting). *A place-sorting* $\Sigma_{\text{K}} = (\mathcal{K}, \Theta, \Phi)$ *over a kind signature* $\mathcal{K}$ *is a* kind sorting *if* $\Theta = \mathcal{P}(\mathcal{K}_{\mathsf{vis}})$ *and* Φ *requires for all bigraphs* G *that:*

K1 *if* $p = G(v)$ *then* $ctrl(v) \in sort(p)$*;*

K2 *if* $p = G(s)$ *then* $sort(s) \subseteq sort(p)$*;*

K3 *if* $sort(v) = \emptyset$*,* v *has no children;*

where p *is a root or node,* s *a site, and* v *is a node.*

We talk about the sort of a node meaning the sort of the control of that node and treat other properties of controls similarly. The sort of a node v is denoted by $sort(v)$ and similarly for roots r and sites s. Given a bigraph G and a node v or site s, $G(v)$ and $G(s)$ denote the parent of the node or site in the bigraph.

The notion of 'multi-nodes' was recently introduced by Milner [27] and allows an ordering of the children of nodes in a bigraph. For example, the $\mathsf{app}^{A \to B, A}$ control in Figure 3 depicts a control modelling a λ application. The function is stored in the left of the node and the argument in the right. This is an example of a 2-node.

The definition of kind sorting presented here is a slight generalisation of our previous definition, paving the way for a proper interpretation of multi-nodes as elements of kind signatures. A node of a bigraph is said to be *exposed* if it is a child of a root, otherwise *hidden*. By restricting the set of interface sorts to $\mathcal{P}(\mathcal{K}_{\mathsf{vis}})$ in the definition above, we force all nodes of invisible controls to be hidden *i.e.* all invisible nodes are the child of some other node. In particular, they can never be children of a root of a redex or reactum in a reaction rule. Invisible controls will allow us (in future work – see Appendix B.1) to define multi-nodes as basic notions of kind signatures. For now, we use a weaker 'implementation' in our encoding of $\Lambda^{\tau}_{\mathrm{sub}}$.

The kind sorting rules are preserved by identities, composition, and tensor product. An interesting identity is $\mathrm{id}_{\langle 1,\emptyset,X\rangle}$. This is semantically as good as a barren root and so maybe $\emptyset$ should not be allowed as an interface sort in the subcategory of hard bigraphs. Kind bigraphs (bigraphs which are kind sorted) have relative pushouts and there is a sorting functor, a forgetful, faithful functor $\mathcal{U}$ from kind bigraphs to pure bigraphs which forgets the sorting of the signature and interfaces and the *vsbl* function. Therefore, kind sorting is an example of place-sorting.

Definition 6 (kind sorted s-category/Brs)**.** *The Σ_{K}-sorted (or kind sorted) bigraphs form a subcategory of $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\mathcal{K},\Phi)$ denoted by $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$. We call $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ a* kind s-category*. If $\mathcal{R}$ is a set of Σ_{K}-sorted reaction rules then $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}},\mathcal{R})$ is a* Σ_{K}-sorted *(or* kind sorted *Brs.*

3.3 Subcategories

In this section, we study subcategories of kind s-categories. Our motivation for studying subcategories of kind s-categories is twofold.

First, we wish to identify subcategories which reflect pushouts. Kind s-categories create RPOs but do not reflect pushouts in general [9]. This latter property may be too strong for some purposes (a weaker property is discussed at the end of this section) but we cannot ignore it completely. The reason why pushouts are not reflected in kind s-categories is that given a bigraph with an arbitrary place graph, there is usually some choice as to how to sort the outer interface. There are thus many commutative squares of bigraphs with the same underlying pure square. If the pure square is a pushout then we at least need the property that all the sorted squares are isomorphic to prove pushout reflection. This is not the case.

This leads us to consider restricting kind s-categories. A natural approach is to identify subcategories where pushouts are reflected. This may be necessary when considering transition systems where the labels are not only generated by IPOs.

Secondly, and more importantly, subcategories may allow a better modelling of some systems or calculi. For example, Milner used a homomorphic subcategory to model finite CCS[1] and we use a partitioned subcategory to model simply typed Λ_{sub}. The use of subcategories may also reduce the set of labels in a transition system which may be required to prove operational correspondence with some calculus and its bigraphical modelling. We describe homomorphic and partitioned subcategories below along with some more which may prove useful in the future.

Definition 7 ((set of) interface/control sorts)**.** *The* (set of) interface sorts *of a subcategory of a kind s-category $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ is the smallest set which contains the sort of any place of any interface in the s-category and is defined by*

$$\left\{\theta \mid \theta \in \vec{\theta}, \langle m, \vec{\theta}, X\rangle \in \mathrm{obj}(\ \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}}))\right\}.$$

The (set of) control sorts *of a kind s-category over signature $\mathcal{K}$ is the set comprised of the union of the sorts of* non-atomic *controls $\mathcal{K}$ and is defined by $\bigcup_{K\in\mathcal{K}} sort(K)$.*

[1] We describe homomorphic sortings as a special case of kind sorting in Appendix D.

Definition 8 (unioned s-category)**.** *A unioned s-category is a full subcategory of a kind s-category* $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *where if* θ *and* θ' *are interface sorts then so is* $\theta \cup \theta'$*.*

Definition 9 (fitting s-category)**.** *A fitting s-category* $\mathcal{F}\mathrm{KB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *is a subcategory of a kind s-category* $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *with the following properties. Let* G *be a bigraph of the subcategory with outer (sorted) interface* I*. Given* $\mathcal{U}(G)$*, the sort of* I *is the smallest choice such that* $\mathcal{U}(G)$ *may be sorted according to the kind signature where* θ *is 'smaller than'* θ' *if* $\theta \subset \theta'$ *or* $|\theta| < |\theta'|$*.*

The key property of fitting s-categories is that we always choose the 'smallest' outer interface and that there is a smallest interface. In particular, no two interfaces of the same cardinality may be used to sort the outer interface of the same bigraph. This means that two bigraphs in the fitting s-category with identical parent maps and inner interfaces must have equal outer interfaces *i.e.* they are in the same homset and are equivalent.

Remark. *From now on, the term bigraph (resp. place graph) usually refers to a fitting bigraph (resp. fitting place graph) of an arbitrary fitting s-category. When we talk of fitting s-categories, it is always in relation to some kind s-category.*

Definition 10 (examples of fitting s-categories)**.** *A fitting s-category is defined by defining the set of sorts allowable for interfaces. The interface sorts must be a subset of* $\mathcal{P}(\mathcal{K}_{\mathsf{vis}})$*. In the following, we do not require that* $\emptyset$ *is an interface sort. A fitting s-category is:*

- (a) meet (s-category) *if for each pair* θ*,* θ' *where* θ *is an interface sort and* θ' *is an interface sort or control sort,* $\theta \cap \theta'$ *is an interface sort;*
- downward closed *if for each interface sort* θ*, any subset of* θ *is an interface sort;*
- controlled *if the set of interface sorts contains the set of control sorts which do not consist entirely of invisible controls;*
- unioned *if the union of any pair of interface sorts is an interface sort;*
- standard *if the set of interfaces sorts is* $\mathcal{P}(\mathcal{K}_{\mathsf{vis}})$*;*
- partitioned *if distinct interface sorts are pairwise disjoint;*
- fully partitioned *if it is partitioned and each visible control is an element of some interface sort;*
- homomorphic *if it is sorted with a homomorphic kind sorting and the interface sorts are exactly the homomorphic groupings (see Appendix D);*

For brevity, we omit the qualifier 'fitting' when referring to (fully) partitioned or homomorphic s-categories.

These definitions identify more types of fitting s-categories than in previous work. The notion we previously called 'fitting bigraphs' [9] is now a particular case of the above definition, the standard fitting s-category. The standard fitting s-category only removes arrows (bigraphs) from the kind s-category. All other fitting s-categories remove objects (interfaces) as well.

There are two flavours of fitting s-category here. The standard fitting s-category is a particular downward closed s-category[2] which are themselves examples of meet s-categories. These s-categories satisfy the property that intersections of pairs of interface sorts always exist. A homomorphic s-category is a special case of a fully partitioned category which is a special case of a

[2]It is also both a particular downward closed and unioned s-category and a downward closed and controlled s-category.

	Subcategory of $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$	Creates RPOs	Reflects pushouts
non-fitting	$\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$	✓ [9]	✗ [9]
non-fitting	unioned	✓	✗
fitting	meet	✗	✓
fitting	downward closed	✗	✓
fitting	– and controlled	✓	✓
fitting	– and unioned	✓	✓
fitting	standard fitting	✓	✓
fitting	partitioned	✓	✓

Figure 2: Proven properties of arbitrary kind s-categories

partitioned category. These s-categories satisfy the property that intersections of pairs of interface sorts only exist when both sorts are equal. An important difference between general partitioned s-categories and homomorphic categories is that the latter also implies restrictions on the sorting of *controls*.

In previous work, we showed that both the standard fitting s-category and any homomorphic s-category reflect pushouts (Milner had previously proved that homomorphic sortings reflect pushouts). These results are generalised in Appendix A. Figure 2 summarises the properties of the various subcategories of an arbitrary kind s-category. A negative mark means that in the general case, this property is not guaranteed. A counterexample for RPO creation of meet and downward closed s-categories is given in the appendices and a counterexample for pushout reflection of unioned s-categories is mentioned.

In this paper, we concentrate on the property of (strong) pushout reflection. Recent work by Bundgaard and Sassone [8] introduces a definition of *weak* pushout reflection which is easier to satisfy and still has positive implications for the labelled transition system based on 'minimal' labels. They focus on IPOs – which underlie much of the transition theory – rather than arbitrary bounds in their definition.

We believe that their results will allow a wider class of subcategories of kind s-categories to have tractable transition systems. In particular, we believe that some non-fitting subcategories, including those mentioned in the figure, may weakly reflect pushouts. A proof would be based on the property that the RPO interface (the centre of the triple) and the IPO interface of RPOs and IPOs of kind sorted bigraphs is minimal. We do not pursue a proof here as we will be further generalising the definition of kind sorting (see Appendix B.1). We will also be standardising our terminology which may improve the communication of the proofs and relate them to recent work by Birkedal, Debois, and Hildebrandt [2]. We will present proofs for the generalisation in future work.

Before we proceed, we will make some observations about the prime product operator.

Remark. *The prime product of two bigraphs in a fitting s-category is defined only when there exists an interface sort which is a superset of the union of the outer interface sorts of the two bigraphs. The prime product of two bigraphs is always defined in the standard fitting s-category. It is defined in a partitioned s-category when all roots of both bigraphs have the same sort.*

4 Simply typed $'\Lambda$BIG

We now revisit Milner's bigraphical model [27] for Λ_{sub}, adding simply typed versions of Λ_{sub} controls to the signature to define $'\Lambda\mathrm{BIG}^{\tau}$, simply typed $'\Lambda$BIG.

We begin in Section 4.1 by defining the signature of $'\Lambda\mathrm{BIG}^{\tau}$ and the kind sorting. The signature is an infinite set – while bigraphs are themselves finite, the definition of a bigraphical reactive system allows a signature based on an infinite set as well as an infinite set of ground reaction rules. We seek a sorting which can properly represent the typing of $\Lambda^{\tau}_{\mathrm{sub}}$. In Section 4.2, we introduce the sorted versions of the reaction rules of $'\Lambda$BIG and identify the fitting s-category in which we will work. All the definitions for $'\Lambda\mathrm{BIG}^{\tau}$ are then collected concisely before we discuss the relationship between the reaction relation of $'\Lambda\mathrm{BIG}^{\tau}$ and $'\Lambda$BIG in the next section.

4.1 Simply typed sorting

The signature for $'\Lambda\mathrm{BIG}^{\tau}$ is based on an infinite set. Instead of introducing typing for bigraphs, what we are essentially doing is taking the typing rules of Figure 1 and using them to generate a corresponding sorted term grammar.

Definition 11 ($'\Lambda\mathrm{BIG}^{\tau}$ controls)**.** *The set of controls $\mathcal{K}^{\tau}$ of $'\Lambda\mathrm{BIG}^{\tau}$ is*

$$\bigcup_{A,B\in\tau} \left\{ \mathsf{lam}^{A\to B} : 1,\ \mathsf{app}^{A\to B,A} : 0,\ \mathsf{var}^{A} : 1,\ \mathsf{sub}^{B,A} : 1,\ \mathsf{def}^{A} : 1,\ 1^{A} : 0,\ 2^{A} : 0,\ \mathsf{D}^{A} : 0 \right\}.$$

The controls $\mathsf{lam}^{A\to B}$ and $\mathsf{sub}^{B,A}$ are binding for all A, B. The other controls are nonbinding. The var^{A} controls are the only atomic controls. All other controls are active.

There is at one control for each type of a λ constructor. $\mathsf{lam}^{A\to B}$ will be used to encode an abstraction of type $A \to B$ where the binding variable of the abstraction has type A, $\mathsf{app}^{A\to B,A}$ will encode applications of type B which take functions of type $A \to B$ and arguments of type A, and var^{A} will encode a variable of type A. There are two controls for each type of explicit substitution. def^{A} will encode a body of substitution of type A in an explicit substitution encoded by $\mathsf{sub}^{B,A}$ of type B.

Controls $1^{A\to B}$ and 2^{A} are respectively used to contain the function and argument part of an application $\mathsf{app}^{A\to B,A}$. Application nodes can then be seen as multi-nodes with an ordered sequence of segments. This idea was proposed by Milner [27]. Similarly, controls 1^{B} and D^{A} are respectively used to contain the target of substitution and body of substitution (def^{A}) part of an explicit subsitution $\mathsf{sub}^{B,A}$

In our encoding, sub is a multi-node. This contrasts with Milner's initial encoding. The reason why we take this approach has to do with the sorting of interfaces discussed later. In order to respect the sorting, we cannot allow an interface sort to contain both a def^{A} control and any other control. We could solve this by defining a $\mathsf{sub}BA$ 'ion' as having two sites, one of which can only contain def^{A} controls and the other which cannot contain any. However, we wish to keep our definitions of ions inner-injective *i.e.* that no two sites are siblings, so we use the multi-node approach to separate the components of sub nodes.

The main controls are depicted in Figure 3, where an application control uses the $1^{A\to B}$ and 2^{A} controls to order its components. The $1^{A\to B}$ and 2^{A} controls are not explicitly shown in the figure but are implied by the sectioning of $1^{A\to B}$. The boxed labels on the holes denote that only controls of type A or B can be placed inside them – we will make this notion more precise when we discuss the fitting s-category in Section 4.2.

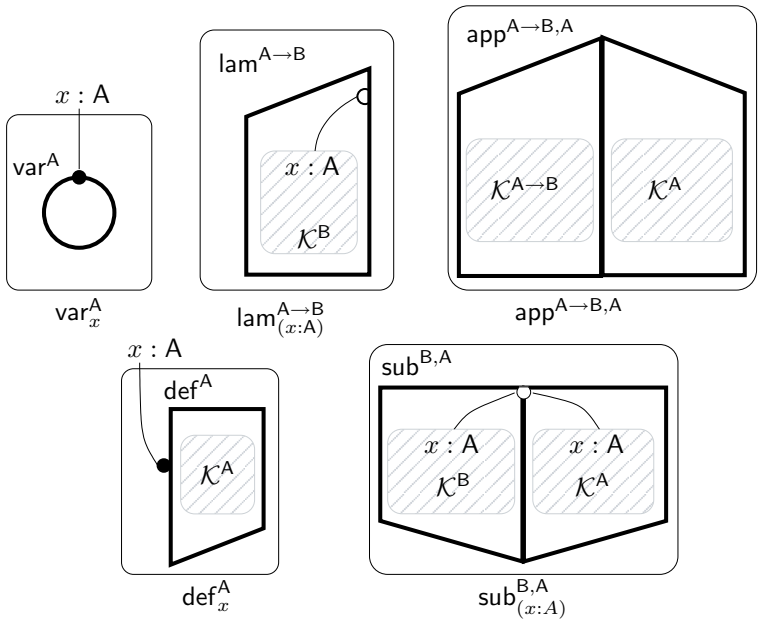


Figure 3: Ion schema for $'\Lambda\text{BIG}^{\tau}$

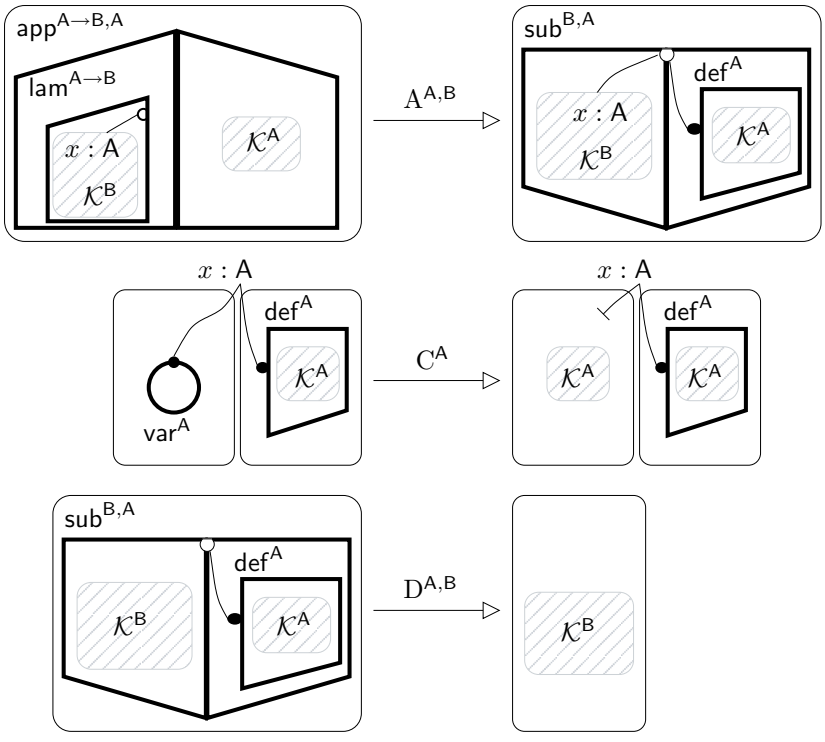


Figure 4: Parametric reaction rule schema for $'\Lambda\text{BIG}^{\tau}$

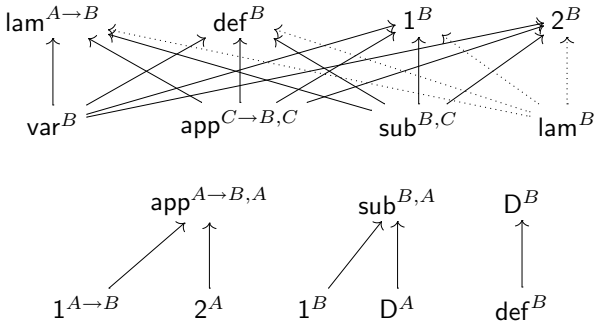


Figure 5: Representation of the simply typed $'\Lambda\text{BIG}^{\tau}$ sorting

Next, we need to introduce a notion of sorting on bigraphs which respects the typing rules. Thus far, we have talked about kind sortings of *pure* bigraphs. $'\Lambda\text{BIG}$ is defined using local bigraphs [27] which allow the modelling of bound names as in the λ-calculus. We also define $'\Lambda\text{BIG}^{\tau}$ as a local bigraphical reactive system. We may still use a kind sorting as this sorting only affects the place graph structure whereas local bigraphs differ from pure bigraphs in the link graph structure. Therefore, among other things, the RPO construction for the place graph structure of kind local bigraphs is the same as for kind bigraphs.

Definition 12 (simply typed $'\Lambda\text{BIG}^{\tau}$ sorting)**.** *$'\Lambda\text{BIG}^{\tau}$ is sorted with a kind sorting on the $'\Lambda\text{BIG}^{\tau}$ signature where the kind function is defined by*

$$
\begin{aligned}
kind(\mathsf{lam}^{A\to G}) = kind(\mathsf{def}^{G}) = kind(1^{G}) = kind(2^{G}) &= \{\mathsf{var}^{G}\} \cup \bigcup_{B\in\tau} \{\mathsf{app}^{B\to G,B}, \mathsf{sub}^{G,B}\} \\
kind(\mathsf{lam}^{A\to\alpha}) = kind(\mathsf{def}^{\alpha}) = kind(1^{\alpha}) = kind(2^{\alpha}) &= \{\mathsf{var}^{\alpha}\} \cup \bigcup_{B\in\tau} \{\mathsf{app}^{B\to\alpha,B}, \mathsf{sub}^{\alpha,B}, \mathsf{lam}^{\alpha}\} \\
kind(\mathsf{app}^{A\to B,A}) &= \{1^{A\to B}, 2^{A}\} \\
kind(\mathsf{sub}^{B,A}) &= \{1^{B}, \mathsf{D}^{A}\} \\
kind(\mathsf{D}^{A}) &= \{\mathsf{def}^{A}\} \\
kind(\mathsf{var}^{A}) &= \emptyset
\end{aligned}
$$

for any A, B, G, α, and the invisible controls are $\bigcup_{A,B\in\tau}\{1^{A}, 2^{A}, \mathsf{D}^{A}\}$.

As a kind sorting is a relation, it can be visualised by a directed graph where nodes of the graph are controls and edges represent the 'can be contained in' relation which arises from a kind signature. Figure 5 depicts a schema for the graph of the sorting above where C is an arbitrary type and the dotted lines are filled when B is a function type.

Note that our example of a multi-node is weaker than Milner's intended proposal – the sorting above allows an $\mathsf{app}^{A\to B,A}$ node to contain many or no 1^{A} and 2^{B} nodes. A more correct notion of multi-node can be achieved by further generalising kind sortings. We consider this in Appendix B.1.

Remark (sorted links)**.** *There is an implied link sorting in our diagrams which we do not concentrate on in our discussions. We present the sorting in Appendix C and prove that it creates RPOs. The sorting condition is that links connect points of the same sort. The sorting is on the link graph structure of local bigraphs and so should combine with kind sorting to maintain RPO creation. In*

the case of $'\Lambda\text{BIG}^\tau$*, set of sorts is* τ *and the sort of the ports of* $\mathsf{lam}^{A\to B}$*,* $\mathsf{sub}^{B,A}$*,* def^A*, and* var^A *controls is* A*.*

So far, we have everything necessary to define a kind s-category of bigraphs. The next question is whether or not we should use a fitting s-category. We have touched upon the fact that kind s-categories do not reflect pushouts. We know that at least some fitting s-categories reflect pushouts and it seems reasonable to try and find one which matches our model.

There is a good reason to use a fitting s-category. We wish for the encoding to respect the typing rules of $\Lambda^\tau_{\text{sub}}$ and so we would like the set of interface sorts to have some relationship to the set of types. Another point is that even though def^A and var^A have the same 'type', it does not make sense from a modelling perspective to include them in the same interface sort as no control which can contain a def^A node can contain a var^A node.

4.2 Sorting the interfaces

We will now identify a fitting s-category of $'\Lambda\text{BIG}^\tau$ bigraphs. This amounts of choosing the sorts of interfaces we want to allow. This is not always a free choice. The task of sorting the interfaces depends somewhat on the reaction rules of the system. Bigraphical reactive rules have the property that the redex and reactum have equal outer interfaces. In order to define a kind sorted Brs, the reaction rules must be well sorted *i.e.* the redex and reactum must have the same sort. This forces the choice of interface sorts to some extent and affects the kind of fitting s-categories which can be defined for some s-category of kind bigraphs.

Example 1 (not all kind reactive systems may be fully partitioned)**.** *Let*

$$\Sigma_{\text{K}} = (\mathcal{K}, \mathsf{vis}^{\mathcal{K}}, \mathcal{P}(\mathcal{K}), \Phi)$$

where $\mathcal{K} = \{A, B, C, D\}$ *with all controls visible and of arity zero. Let* Φ *define* A *and* B *as atomic and the kinds of* C *and* D *as* $\{A, B\}$ *and* $\{B\}$ *respectively. Define a ground reaction rule* (A, B) *with interface* $\epsilon \to \langle 1, \{\{A, B\}\}, \emptyset\rangle$*, transforming an* A *node to a* B *node. Let* (R, R') *be a second parametric reaction rule where the redex contains a* D *node containing a site of sort* B *and this site is discarded in the reactum.*

A fully partitioned s-category based on this signature and ruleset must both have a superset of $\{A, B\}$ *as a sort to match the ground rule and a subset of* $\{B\}$ *as a sort to match the parametric rule. The only valid choice, letting the inner sort of the parametric redex be* $\emptyset$*, destroys the intended meaning of the parametric rule.*

We would like to use a fully partitioned s-category to model $\Lambda^\tau_{\text{sub}}$ so that the sorts correspond in some way with the types. If we look at the schema for $'\Lambda\text{BIG}^\tau$ sorting in Figure 5, we see that $'\Lambda\text{BIG}^\tau$ has a similar property as in the example above – $\mathsf{app}^{A\to B,A}$ can contain a $\mathsf{1}^{A\to B}$ and a $\mathsf{2}^A$ whereas $\mathsf{sub}^{A\to B,A}$ can only contain the former. Further, an $\mathsf{app}^{A\to B,A}$ node can contain $\mathsf{1}^{A\to B}$ and $\mathsf{2}^A$ nodes whereas an $\mathsf{app}^{A\to C,A}$ node can contain $\mathsf{1}^{A\to C}$ and $\mathsf{2}^A$ nodes. It would appear that we need different interface sorts for each $\mathsf{1}^A$, $\mathsf{2}^A$, and D^A control which is not particularly desirable. However, these controls are invisible and are not allowed to be elements of interface sorts which fixes this problem. Invisible controls cannot affect whether a certain fitting s-category may be found and in $'\Lambda\text{BIG}^\tau$ they are merely a means of proving some syntactic sugar – they are merely used as a means of adding extra structure to the place graph. Leaving invisible controls out of the interface sorts ensures that these controls are always tied to some lam or sub control in the bigraphs of the system *i.e.* they cannot exist on their own.

The reaction rules for $'\Lambda\text{BIG}^\tau$ are depicted in Figure 4 (the meaning of the type labels on the sites will be explained later). This figure is a rule schema – for every pair (A, B) of types there are

corresponding rules $\mathrm{A}^{\mathrm{A,B}}, \mathrm{C}^{\mathrm{A}}, \mathrm{C}^{\mathrm{B}}$, and $\mathrm{D}^{\mathrm{A,B}}$. If we forget the simple type annotations and $\mathsf{sub}^{B,A}$ being a multi-node, they depict Milner's rules for untyped Λ_{sub}.

Ideally, we want a fitting s-category where the interface sorts are such that for any bigraph which correctly models a $\Lambda^{\tau}_{\mathrm{sub}}$ term:

1. there is exactly one choice of outer interface sort, and
2. the interface is related somehow to the type of the $\Lambda^{\tau}_{\mathrm{sub}}$ term.

The first point implies that the interface sorts should be minimal in some sense and, with the second point, this suggests a solution. We will examine the reaction rules to demonstrate this.

Consider the rule $\mathrm{D}^{\mathrm{A,B}}$. The redex must have a sort containing $\mathsf{sub}^{B,A}$. According to the typing system of $\Lambda^{\tau}_{\mathrm{sub}}$ and our sorting for $'\Lambda\mathrm{BIG}^{\tau}$, the site which persists in the reactum is able to contain any var, app, lam, or sub nodes of 'type' B. Therefore, an appropriate sort for the rule seems to be the set of all such nodes, or $\Theta(1^B)$. It is sometimes possible for a $\mathrm{D}^{\mathrm{A,B}}$ reaction to follow a $\mathrm{A}^{\mathrm{A,B}}$ reaction in $\Lambda^{\tau}_{\mathrm{sub}}$. Therefore, $\Theta(1^B)$ also seems an appropriate sort for $\mathrm{A}^{\mathrm{A,B}}$.

This suggests a sorting – we base the allowed sorts for interfaces on the simple types of $\Lambda^{\tau}_{\mathrm{sub}}$ and use a partitioned s-category. We need to take some care. No interface which contains a def node should be able to contain any var, app, lam, or sub controls. Otherwise, under the rules of the sorting, we would end up with a subcategory of ground bigraphs and identities as no ion can be formed with a site of that sort. We therefore define the set of interface sorts as follows. For each type $A \in \tau$:

1. There is a sort which contains all var, app, lam, and sub controls which model a $\Lambda^{\tau}_{\mathrm{sub}}$ construct of that type (for example, $\Theta(1^A)$ of Definition 12).
2. There is a sort $\{\mathsf{def}^A\}$.

Note how this definition resembles the diagram in Figure 5 if we ignore the invisible controls. This choice of interface sorts defines a fully partitioned s-category which is not homomorphic.

4.3 $'\Lambda\mathrm{BIG}^{\tau}$

Before continuing, we will gather all the definitions together.

Definition 13 (typed subsets of $\mathcal{K}^{\tau}$)**.** *For any ground type G and function type α we define the following.*

$$\begin{aligned} \mathcal{K}^{G} &= \{\mathsf{var}^{G}\} \cup \bigcup_{B \in \tau} \{\mathsf{app}^{B\to G,B}, \mathsf{sub}^{G,B}\} \\ \mathcal{K}^{\alpha} &= \{\mathsf{var}^{\alpha}\} \cup \bigcup_{B \in \tau} \{\mathsf{app}^{B\to \alpha,B}, \mathsf{sub}^{\alpha,B}, \mathsf{lam}^{\alpha}\} \end{aligned}$$

Definition ($'\Lambda\mathrm{BIG}^{\tau}$ signature)**.** *The set of controls of $'\Lambda\mathrm{BIG}^{\tau}$ is*

$$\mathcal{K}^{\tau} = \bigcup_{A,B \in \tau} \{\mathsf{lam}^{A\to B} : 1,\ \mathsf{app}^{A\to B,A} : 0,\ \mathsf{var}^{A} : 1,\ \mathsf{sub}^{B,A} : 1,\ \mathsf{def}^{A} : 1,\ 1^{A} : 0,\ 2^{A} : 0,\ \mathrm{D}^{A} : 0\}.$$

The controls $\mathsf{lam}^{A\to B}$ and $\mathsf{sub}^{B,A}$ are binding with arity 1 for all A, B. var and def controls have arity 1 are are nonbinding. All other controls are nonbinding with arity 0. The var^{A} controls are

the only atomic controls. All other controls are active. All $1, 2$, *and* D *nodes are invisible, all others are visible. The kind function is defined by:*

$$\begin{aligned} kind(\mathsf{lam}^{B\to A}) = kind(\mathsf{def}^A) = kind(1^A) = kind(2^A) &= \mathcal{K}^A \\ kind(\mathsf{app}^{A\to B,A}) &= \{1^{A\to B}, 2^A\} \\ kind(\mathsf{sub}^{B,A}) &= \{1^B, \mathsf{D}^A\} \\ kind(\mathsf{D}^A) &= \{\mathsf{def}^A\} \\ kind(\mathsf{var}^A) &= \emptyset \end{aligned}$$

for any A, B.

We now explain the labelling of the sites in the reaction rules of Figure 4. For any type A, if a site is labelled A then it has the sort $\mathcal{K}^A$. Note also that the redex and reactum of these reaction rules are both well sorted and outer interfaces have equal sorts.

Definition 14 ($'\Lambda\text{BIG}^\tau$)**.** *$'\Lambda\text{BIG}^\tau$ is the reactive system over the fully partitioned s-category of $\mathcal{B}\text{IG}_\text{h}(\Sigma_\text{K})$, $\Sigma_\text{K} = (\mathcal{K}^\tau, vsbl^\tau, \mathcal{P}(\mathcal{K}^\tau), \Phi^\tau)$ where the set of interface sorts is $\{\mathsf{def}^A \mid A \in \tau\} \cup \{\mathcal{K}^A \mid A \in \tau\}$ and the reaction rules are as depicted in Figure 4.*

When a bigraph G of $'\Lambda\text{BIG}^\tau$ is prime and the root has one child, we write $sort(G)$ for the sort of the root.

5 Encoding Λ^τ_sub in $'\Lambda\text{BIG}^\tau$

We can now encode Λ^τ_sub in $'\Lambda\text{BIG}^\tau$. The use of a partitioned subcategory is useful in forcing the sort of an encoding of a term t to match the type of t under some environment Γ. By defining $'\Lambda\text{BIG}^\tau$ similarly to $'\Lambda\text{BIG}$ we can also easily recover the latter through a partial functor. Finally, we use properties of Λ_sub to infer properties of Λ^τ_sub and the image of Λ^τ_sub in the encoding.

Note that while we use a partitioned subcategory and so have pushout reflection, the reaction rules in $'\Lambda\text{BIG}^\tau$ are not all prime so that the Theorem 2.2 does not immediately apply.

5.1 The encoding

We can now model the simply typed λ-calculus with an explicit substitution bigraphical system. We do this by giving a translation of Λ^τ_sub into $'\Lambda\text{BIG}^\tau$, following Milner's encoding of Λ_sub. The translation $[\![t]\!]_\Gamma$ of a term t is indexed by Γ, an environment which can type t in Λ_sub.

Our encoding uses the extension operator $\oplus$ on bigraphs, which adds localised names to a bigraph. We implicitly generalise this operator to our link sorted interfaces by forming the union of the environments (which always have disjoint domains). The extension operation on bigraphs is defined as usual and preserves our link sorting.

Given two environments Γ and Γ', if $\text{dom}(\Gamma) \subseteq \text{dom}(\Gamma')$ and both enviroments are compatible, we overload notation and write $\Gamma \subseteq \Gamma'$. The encoding of a derivation $\Gamma \vdash t : A$ is indexed with an environment Γ' where $\Gamma \subseteq \Gamma'$ and $\mathsf{BV}(t) \cap \text{dom}(\Gamma') = \emptyset$. This indexing is required in order to prove that Λ^τ_sub reduction matches $'\Lambda\text{BIG}^\tau$ reaction as the former loses free variables whereas the latter does not.

Definition 15 (encoding type derivations as bigraphs)**.** *The encoding $[\![-]\!]_{\Gamma'}$ which takes a Λ^τ_sub derivation $\Gamma \vdash t : A$ with $\Gamma \subseteq \Gamma'$ to a prime, ground bigraph $G : \epsilon \to \{1, \mathcal{K}^A, \text{dom}(\Gamma'), \Gamma'\}$ is defined inductively on the inference of $\Gamma \vdash t : A$ and presented in Figure 6.*

The encoding preserves types in some way; if $\Gamma \vdash t : A$ then $sort[\![\Gamma \vdash t : A]\!]_{\Gamma'} = \mathcal{K}^A$.

Derivation	Encoding with index Γ'
$\Gamma, x : A \vdash x : A$	$\mathsf{var}^{A}_{x:A} \oplus \Gamma'$
$\Gamma \vdash t\ u : A$	$\mathsf{app}^{B\to A,B} \oplus (\mathrm{id}_{\Gamma'} \mid \mathrm{id}_{\Gamma'})$ $\circ\, ([\![\Gamma \vdash t : B \to A]\!]_{\Gamma'} \parallel [\![\Gamma \vdash u : B]\!]_{\Gamma'})$
$\Gamma \vdash t[x/u] : A$	$\mathsf{sub}^{A,B}_{(x:B)} \oplus (\mathrm{id}_{\Gamma'} \mid \mathrm{id}_{\Gamma'})$ $\circ\, \big([\![\Gamma, x : B \vdash t : A]\!]_{\Gamma',x:B}(\mathsf{def}^{B}_{x:B} \oplus \mathrm{id}_{\Gamma'})[\![\Gamma \vdash u : B]\!]_{\Gamma'}\big)$
$\Gamma \vdash \lambda x.t : A \to B$	$(\mathsf{lam}^{A\to B}_{(x:A)} \oplus \mathrm{id}_{\Gamma'})[\![\Gamma, x : A \vdash t : B]\!]_{\Gamma',x:A}$

Figure 6: Encoding of $\Lambda^{\tau}_{\mathrm{sub}}$ terms into $'\Lambda\mathrm{BIG}^{\tau}$

Proposition 3. *The encoding $[\![\Gamma \vdash t : A]\!]_{\Gamma'}$ of a $\Lambda^{\tau}_{\mathrm{sub}}$ derivation is well-sorted with outer kind sort $\mathcal{K}^{A}$ and link sort Γ'.*

Proof. By induction on the derivation of t. We explain how the derivations of subterms in the inductive cases are valid, then show that the encoding respects the sorting. Note that all $'\Lambda\mathrm{BIG}^{\tau}$-ions are sorted and that composition preserves kind sorting. We can see from the definition that the outer link sort is Γ' in all cases.

1. $[\![\Gamma, x : A \vdash x : A]\!]_{\Gamma'}$.

   All ions are sorted.

2. $[\![\Gamma \vdash t\ u : A]\!]_{\Gamma'}$.

   The last step of the derivation $\Gamma \vdash t\ u : A$ must have been an application of $\mathtt{app}$. Therefore, $\Gamma \vdash t : B \to A$ and $\Gamma \vdash u : B$ for some B.

   The encodings $[\![\Gamma \vdash t : B \to A]\!]_{\Gamma'}$ and $[\![\Gamma \vdash u : B]\!]_{\Gamma'}$ have outer sorts $\mathcal{K}^{B\to A}$ and $\mathcal{K}^{B}$ respectively and are sorted by the i.h.. Hence, the entire term is sorted.

3. $[\![\Gamma \vdash t[x/u] : A]\!]_{\Gamma'}$.

   The last step of the derivation $\Gamma \vdash t[x/u] : A$ must have been an application of $\mathtt{subs}$. Therefore, $\Gamma, x : B \vdash t : A$ and $\Gamma \vdash u : B$ for some B.

   The encodings $[\![\Gamma, x : B \vdash t : A]\!]_{\Gamma'}$ and $[\![\Gamma \vdash u : B]\!]_{\Gamma'}$ have outer sorts $\mathcal{K}^{A}$ and $\mathcal{K}^{B}$ respectively and are sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the sub node is linked to an inner name x of the same sort in the composition.

4. $[\![\Gamma \vdash \lambda x.t : A \to B]\!]_{\Gamma'}$.

   The last step of the derivation $\Gamma \vdash \lambda x.t : A \to B$ must have been an application of $\mathtt{abs}$. Therefore, $\Gamma, x : A \vdash t : B$.

   The encoding $[\![\Gamma, x : A \vdash t : B]\!]_{\Gamma'}$ has outer sort $\mathcal{K}^{B}$ and is sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the lam node is linked to an inner name x of the same sort in the composition.

□

The following proposition follows by induction where $\lambda x.t \equiv_\alpha \lambda y.t\{\,y/x\,\}, y \notin \mathsf{FV}(t)$ and $t[x/u] \equiv_\alpha t\{\,y/x\,\}[y/u], y \notin \mathsf{FV}(t)$ are the non-trivial cases.

Proposition 4. *If $t \equiv_\alpha u$ and $\mathsf{FV}(t) \subseteq X$ then $[\![t]\!]_{X,\Gamma} \mathrel{\dot{=}} [\![u]\!]_{X,\Gamma}$.*

Similarly to $'\Lambda\text{BIG}$ we have the following except that now each of the three rules for Λ_{sub} is matched by an infinite number of $'\Lambda\text{BIG}^\tau$ rules. However, only the corresponding rule with the correct type may be applied.

Proposition 5 (reaction matches reduction)**.** *$[\![t]\!]_{X,\Gamma} \rightharpoonup g$ if and only if $t \longrightarrow_{\text{ACD}} t'$ for some t' such that $[\![t']\!]_{X,\Gamma} \mathrel{\dot{=}} g$.*

5.2 Recovering Λ_{sub}

$'\Lambda\text{BIG}^\tau$ was built by sorting $'\Lambda\text{BIG}$. We can introduce a forgetful (but not faithful) partial functor between the two systems.

Definition 16 ($\mathcal{U}^\tau$)**.** *We define the partial functor $\mathcal{U}^\tau :' \Lambda\text{BIG}^\tau \to' \Lambda\text{BIG}$ as follows. On objects, $\mathcal{U}^\tau(I) = \mathcal{U}(I)$ is the* underlying (local) interface *of I, forgetting kinds of roots and types of names. On ground arrows, $\mathcal{U}^\tau(G)$ is defined as G where the superscripts of controls are omitted and the children $\mathsf{1}^B$ and D^A of all $\mathsf{sub}^{B,A}$ controls are forgotten. $\mathcal{U}^\tau(G)$ is called the* underlying bigraph *of G. Given a ground reaction rule of $'\Lambda\text{BIG}^\tau$, the underlying ground reaction rule is defined as the pair of underlying redex and underlying reactum. The underlying ground reaction rules are exactly the reaction rules of $'\Lambda\text{BIG}$.*

The partial functor is defined on ground terms. We can therefore compose it with the encoding $[\![]\!]_{X,\Gamma}$ from $\Lambda^\tau_{\text{sub}}$ to $'\Lambda\text{BIG}^\tau$ (we omit the side-conditions below).

$$
\begin{aligned}
\mathcal{U}^\tau[\![x]\!]_{X \uplus x:A,\Gamma} &\stackrel{\text{def}}{=} \mathsf{var}_x \oplus X \\
\mathcal{U}^\tau[\![\lambda x : A.t]\!]_{X,\Gamma} &\stackrel{\text{def}}{=} (\mathsf{lam}_{(x)} \oplus \mathrm{id}_X)\mathcal{U}^\tau[\![t]\!]_{X \uplus x:A,(\Gamma,x:A)} \\
\mathcal{U}^\tau[\![t\ u]\!]_{X,\Gamma} &\stackrel{\text{def}}{=} (\mathsf{app} \oplus (\mathrm{id}_X | \mathrm{id}_X))(\mathcal{U}^\tau[\![t]\!]_{X,\Gamma} \parallel \mathcal{U}^\tau[\![u]\!]_{X,\Gamma}) \\
\mathcal{U}^\tau[\![t[x/u]]\!]_{X,\Gamma} &\stackrel{\text{def}}{=} (\mathsf{sub}_{(x)} \oplus \mathrm{id}_X)(\mathcal{U}^\tau[\![t]\!]_{X \uplus x:A,(\Gamma,x:A)} \mid (\mathsf{def}_x \oplus \mathrm{id}_X)\mathcal{U}^\tau[\![u]\!]_{X,\Gamma})
\end{aligned}
$$

For a term t of $\Lambda^\tau_{\text{sub}}$, let $\mathcal{U}(t)$ denote the untyped term of Λ_{sub} which forgets the types of abstractions.

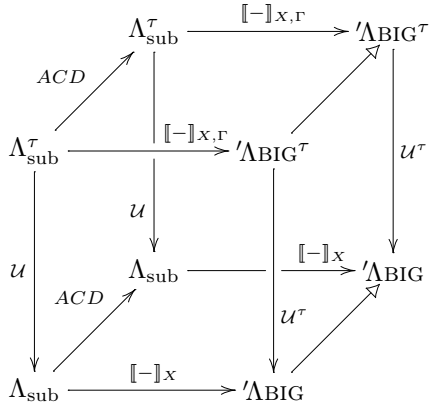
Proposition 6. $\mathcal{U}^\tau \circ [\![t]\!]_{X,\Gamma} = [\![\mathcal{U}(t)]\!]_X$

Proof. Induct over the structure of t. □

Proposition 7. *$'\Lambda\text{BIG}$ reaction on $[\![-]\!]_{X,\Gamma}$-images strongly simulates $'\Lambda\text{BIG}$ reaction on $[\![-]\!]_X$-images through $\mathcal{U}^\tau$.*

Proof. The proof follows by the "commutative diagram" below. Proposition 6 proves the squares, Proposition 5 proves the 'top face' of the cube, and the matching of Λ_{sub} reduction by $'\Lambda\text{BIG}$

reaction [27] proves the 'bottom face.'



□

This enables us to reflect some properties of $'\Lambda\text{BIG}$ back through $\mathcal{U}^\tau$.

5.3 Properties of $'\Lambda\text{BIG}^\tau$

We now use previous results for Λ_{sub} to prove properties of $\Lambda^\tau_{\text{sub}}$ and $'\Lambda\text{BIG}^\tau$.

Proposition 8. *$\Lambda^\tau_{\text{sub}}$ can simulate β-reduction, is confluent on terms without metavariables, preserves strong normalisation of β-reduction, and has full composition of substitutions. $'\Lambda\text{BIG}^\tau$ has these properties on $[\![-]\!]_{X,\Gamma}$-images of $\Lambda^\tau_{\text{sub}}$ terms.*

Proof. These properties have been proven for Λ_{sub} [10]. $\Lambda^\tau_{\text{sub}}$ inherits the properties of Λ_{sub}. $'\Lambda\text{BIG}^\tau$ gains these properties by Proposition 5. □

Proposition 9. *$\Lambda^\tau_{\text{sub}}$ is strongly normalising.*

Proof. We use Herbelin's approach [15] and Kesner's translation [18].

The translation from Λ_{sub} terms to λ-terms is:

$$\begin{aligned} C(x) &= x \\ C(t\ u) &= C(t)C(u) \\ C(\lambda x.t) &= \lambda x.C(t) \\ C(t[x/u]) &= (\lambda x.C(t))C(u). \end{aligned}$$

The translation reverts all explicit substitutions to β-redexes. We have $C(t) \longrightarrow_{\text{B}}^* t$ for all t. By inducting over the number of explicit substitutions in t, and using the fact that $\longrightarrow_{\text{B}}$ preserves types, we can prove that if $\Gamma \vdash t : A$ then $\Gamma \vdash C(t) : A$. Moreover, the derivation of $\Gamma \vdash C(t) : A$ uses the rules of the simply typed λ-calculus. Therefore, $C(t)$ is strongly normalising for β-reduction [1]. As Λ_{sub} has PSN, we conclude that t is strongly normalising. □

We have the following corollary of Propositions 5 and 9.

Corollary 10. *$'\Lambda\text{BIG}^\tau$ is strongly normalising on $[\![-]\!]_{X,\Gamma}$-images of $\Lambda^\tau_{\text{sub}}$ terms.*

6 Conclusions

We have presented a bigraphical encoding of an explicit substitution calculus where typable terms do terminate. This provides a significant theoretical application for kind sortings. We have also identified subcategories of kind bigraphs which retain some important properties of the transition theory of pure bigraphs and which we hope will be useful in future applications. The proofs of retention generalise previous results by identifying a wider class of these subcategories which reflect pushouts.

The notion of invisible controls pave the way towards modelling multi-nodes with kind bigraphs and we continue in this direction in Appendix B.1.

We can make some observations about kind s-categories. They are very free sorted s-categories *i.e.* allowing the sort of an interface to be *any* subset of a signature $\mathcal{K}$ which satisfies the kind rule is not very restrictive. This has benefits and problems associated with it. The benefit is that some very expressive reaction rules [9] can be written for these systems. The downside is that they do not seem suitable for modelling sorted systems such as finite CCS or the simply typed λ-calculus. For the latter applications, we have seen that subcategories have proven useful.

It seems that when using kind bigraphs, we can choose either to: i) have very expressive reaction rules, which seem useful to model 'abstract' reactive systems; or else to ii) restrict attention to subcategories, which seems to be necessary in order to model certain calculi, and lose some expressivity. The trade-off between these choices is worth investigating.

7 Further and related work

A better model When introducing $'\Lambda\text{BIG}^{\tau}$, we opted for the closest match to $'\Lambda\text{BIG}$ so that we could easily (partially) recover $'\Lambda\text{BIG}$ through the functor $\mathcal{U}^{\tau}$. However, our design does not use kind sorting to its best when modelling explicit substitutions. It may be better to modelling explicit substitutions with a triple of controls $\mathsf{sub}, \mathsf{U}, \mathsf{D}$ instead of with $\mathsf{sub}, \mathsf{1}, \mathsf{D}$, and def. The new sub would have no ports, U would model the left-hand side of sub and would bind the variable of the explicit substitution. def controls seem redundant in $'\Lambda\text{BIG}^{\tau}$ as the D control models the right-hand side of sub.

Sorting There are strong similarities between kind sorting and the definition of sorting for the polyadic π-calculus [23]. We use the set $\mathcal{K}$ as the set of 'subject sorts' and subsets of $\mathcal{K}$ as the set of 'object sorts.' In the polyadic π-calculus, the set of objects sorts is the set of sequences of subject sorts. The definition of kind sorting with capacities in the appendices is closer to the polyadic π-calculus approach.

Kind sortings are a particular example of Milner's definition of place sorting [26] and generalise homomorphic sortings which were used to model finite CCS. Sorting and binding disciplines for link graphs have been studied by Jensen, Milner, and Leifer [16, 21] to model variants of the π-calculus, Petri nets, and arithmetic nets. Work on binding led to local bigraphs [24], used to model the λ-calculus. The definition of local bigraphs has been recently simplified by Milner and we have used that definition here.

Birkedal et al. have proposed a method of modelling context-aware systems by describing a single bigraphical system as a combination of three smaller component systems where each component represents a different view of the world and agents interact with a context via a shared proxy [4, 3]. To ensure compositional safety, they using a particular example of their rigid control-sortings (which create RPOs).

Their definition of rigid control-sorting has similarities with a proposed variant of kind sorting called stronger kind sorting [9]. If differs from kind sorting in that the containment relationship

is between a root and the nodes below the root rather than the parent-child relationship of kind sortings. For this reason, neither sorting can generalise the other except in trivial cases. However, it seems likely that a combination of both approaches – where the sorting constrains containment according to both roots and the parent-child relationship – may be useful. We present such a combination in Appendix B.2 and prove that it creates RPOs.

Birkedal, Debois, and Hildebrandt [2] have introduced a general theory of sorting for reactive systems over categories which will hopefully be extended to cover precategories and 2-categories – bigraphs are defined in the former setting and the latter has been studied as an alternative setting for reactive systems [31]. They introduce a sorting, *predicate sorting*, which generalises any sorting where the sorting condition is retained by composition/decomposition of bigraphs and prove predicate sorting creates RPOs. The relationship between kind sortings and predicate sortings *i.e.* can kind sortings be couched as predicate sortings, deserves attention. So too does the expressivity of their sorted reaction relations with respect to guaranteed *absence*, a notion we explored previously for kind sortings [9]. We briefly relate some of their terminology with ours in Appendix A.1.

Bundgaard and Hildebrandt's bigraphical presentation of the Homer calculus [6, 7] is a presentation of a typed version of Homer with explicit substitutions, Homerσ. However, the type of a Homer term in this case is a set of names which contains the free names[3]of the term. The typing is used to ensure that reduction in Homerσ does not lose free names so that reduction can match reaction in the bigraphical model (where the notion of (outer) interface preservation is implicit in the dynamics). This differs from the typing we are trying to model.

Recently, Bundgaard and Sassone presented a typed polyadic pi-calculus in bigraphs [8] where the type system was based on Pierce and Sangiorgi's capability types [28] with subsorting. They introduce a new link sorting (called subsorting) which represents the first step in a theory of link-subtyping with binding for bigraphs. The approach differs from previous work as the edges of bigraphs are sorted rather than the ports. The sort of a node is then derived from the sorts of the links to which it is connected. As a result, their presentation only uses one control to model an input/output prefix of some message length. In comparison, this paper has introduced a place-sorting where the controls are sorted and where we have one control per type per lambda constructor.

Transition theory We have concentrated on proving (strong) pushout reflection here where any bound whose pure image is a pushout must be a pushout. This may be useful when considering the full transition system where all bounds involving redexes play a part in the labelled transition system. However, results by Bundgaard and Sassone [8] have shown that proving this property for IPOs alone still allows a tractable transition system. We believe that this weaker property holds for the non-fitting (sub)categories in Figure 2, and could be readily shown, using the notion of *fitting bound* [9] but defer this work. Fitting bounds lie between arbitrary bounds and IPOs and seem to be the correct basis for a 'full' transition system for kind bigraphs. Bisimilarity and adequacy theorems for this transition system should be investigated although in practice, the transition system based on IPOs is more likely to be useful.

We have also concentrated on proving RPO creation and pushout reflection of *composite functors* from a fitting subcategory to a sorted s-category to pure bigraphs. It may be simpler to merely prove the property on the inclusion functor from the subcategory to the sorted s-category. It would seem that this is enough to prove that the composite functor has the required property. However, not all such inclusion functors $\mathcal{I}$ create RPOs *e.g.* if the RPO interface in the full s-category is not an interface sort then the RPO cannot have an $\mathcal{I}$-preimage and hence $\mathcal{I}$ would not create RPOs. However, in some of these cases it may still be that the composite $\mathcal{UI}$ does.

Further typing We have modelled the simply typed λ calculus with bigraphs using a relatively simple sorting. This raises the questions of whether a simpler sorting would suffice and what the limitation of kind sorting is with respect to modelling type systems. The latter question could be investigated by attempting to model more simple types (products, coproducts), intersection types [12], or recursion, although we believe that kind sorting can only take us so far. Examining the encodings of typed λ calculi in the π-calculus [32, 30] may get us further.

The set of strongly normalising terms of Λ_{sub} does not yet have a neat characterisation. Typing may provide this characterisation as intersection types have for the λ-calculus [12] and calculi of explicit substitutions [22].

Explicit substitution calculi and proof-nets Milner's Λ_{sub} is confluent on ground terms, preserves strong normalisation of β-reduction, has full composition of substitutions [10], and has a strongly normalising typed variant. It differs from most explicit substitution calculi as it does not propagate explicit substitutions through terms. Besides being a well-behaved model of the λ-calculus, is it of interest for the explicit substitution community? We do not have any answers, intuitions, or expertise in this direction.

The connection between explicit substitution calculi with distributive rules for substitution and cut elimination in proof-nets has provided insights into both areas of research. It has yielded normalisation proofs for existing explicit substitution calculi [13, 14] as well as aided the development of new calculi with some or all of the properties of preservation of strong normalisation (PSN), open confluence, and full composition of substitutions [19, 18].

PSN for Λ_{sub} was proved by simulating reduction in a modified version of one of these later calculi [10]. However, the proof itself does not yield much insight into the relationship between Λ_{sub} and proof-nets. It has been suggested to us that this relationship should be explored.

Applications Conforti, Macedonio, and Sassone have considered bigraphical models of XML data whose document order is irrelevant [11]. They then applied their spatial logic, inspired by bigraphs, on the bigraphs to describe and reason about XML data. They remark that an extension of bigraphs with a notion of "ordered locality" could be used to model XML data whose document order is relevant. Our description of Milner's multi-nodes in a kind sorting with capacities (see Appendix B.1) extends bigraphs in this manner.

Controls in binding signatures have a number of binding ports and a number of non-binding ports. With local bigraphs [27], a control either has binding ports or non-binding ports but not both. Milner shows how this new approach can encode the old approach by using a sorting and a pair of controls, one binding, one non-binding. The place graph component of that sorting can be represent by a kind sorting with capacities where we would also define the inner binding node as invisible.

Some other suggestions are described in Appendices B.1, B.3, and B.4.

Acknowledgements

The details of how this work originated are a bit hazy but Milner's idea of multi-nodes seemed a crucial step. It does not seem clear how to use kind sorting to model simply typed Λ_{sub} without this notion.

Delia Kesner pointed me to her translation from Λ_{sub} terms to λ-terms which is more elegant than the method we initially suggested. Comments from both Malcolm Dowse and Edsko de Vries were very helpful. Previous comments from Søren Debois led to a simpler definition of kind sorting.

References

[1] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition edition, 1984.

[2] L. Birkedal, S. Debois, and T. Hildebrandt. Sortings for reactive systems. In Christel Baier and Holger Hermanns, editors, *Proceedings of CONCUR 2006*, August 2006. To appear.

[3] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. Technical Report 74, IT University of Copenhagen, Rued Langgards Vej 7, DK-2300 Copenhagen V, November 2005. ISBN: 87-7949-110-3.

[4] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. In *Foundations of Software Science and Computation Structures (FOSSACS) 2006*, number 3921 in Lecture Notes in Computer Science, March 2006.

[5] Roel Bloo and Kristoffer Høgsbro Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN-95: Computer Science in the Netherlands*, November 1995.

[6] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. Technical Report TR-2005-70, IT-Universitetet, København, September 2005.

[7] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Reiko Heckel, Barbara König, and Arend Rensink, editors, *Proceedings of GT-VC '05 (Graph Transformation for Verification and Concurrency)*, number 05–34 in CTIT Technical Reports. Centre for Telematics and Information Technology, University of Twente, 2005.

[8] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming 2006*, pages 1–12, 2006.

[9] Shane Ó Conchúir. Kind bigraphs – static theory. Technical Report TCD-CS-2005-36, Trinity College Dublin, May 2005.

[10] Shane Ó Conchúir. Λ_{sub} as an explicit substitution calculus. Technical Report TR-2006-95, IT-Universitetet, København, September 2006.

[11] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Bigraphical logics for xml. In Andrea Calì, Diego Calvanese, Enrico Franconi, Maurizio Lenzerini, and Letizia Tanca, editors, *SEBD*, pages 392–399, 2005.

[12] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.

[13] Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 35–46, Warsaw, July 1997.

[14] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409–450, 2003.

[15] Hugo Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes.* Thèse d'université, Université Paris 7, Janvier 1995.

[16] Ole Høgh Jensen and Robin Milner. Bigraphs and transitions. In *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 38–49, New York, NY, USA, 2003. ACM Press.

[17] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, Computer Laboratory, University of Cambridge, February 2004.

[18] Delia Kesner. The theory of explicit substitutions revisited. Available at "*http://www.pps.jussieu.fr/∼kesner/papers/*".

[19] Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2005.

[20] F. William Lawvere and Stephen H. Schanuel. *Conceptual Mathematics.* Cambridge University Press, 1998. A first introduction to categories.

[21] James J. Leifer and Robin Milner. Transition systems, link graphs and petri nets. Technical Report UCAM-CL-TR-598, Computer Laboratory, University of Cambridge, August 2004.

[22] Stéphane Lengrand, Pierre Lescanne, Dan Dougherty, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[23] Robin Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[24] Robin Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, Computer Laboratory, University of Cambridge, September 2004.

[25] Robin Milner. Pure bigraphs. Technical Report UCAM-CL-TR-614, Computer Laboratory, University of Cambridge, 2005.

[26] Robin Milner. Pure bigraphs: structure and dynamics. Technical Report to appear, Computer Laboratory, University of Cambridge, 2005.

[27] Robin Milner. Local bigraphs, confluence and λ-calculus. Technical Report to appear, Computer Laboratory, University of Cambridge, 2006.

[28] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[29] Kristoffer Høgsbro Rose. Explicit substitution – tutorial & survey, 1996.

[30] Davide Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes.* Cambridge University Press, New York, NY, USA, 2001.

[31] Vladimiro Sassone and Pawel Sobocinski. Deriving bisimulation congruences using 2-categories. *Nord. J. Comput.*, 10(2):163–, 2003.

[32] David N. Turner. *The Polymorphic Pi-calculus: Theory and Implementation.* PhD thesis, University of Edinburgh, 1996.

A Proofs for section 3.2

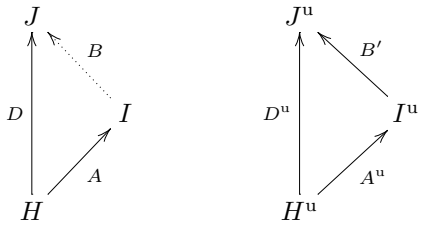
This section contains some proofs identifying fitting s-categories which create RPOs or reflect pushouts. These require different properties. For RPO creation, we need to be able to build an RPO interface (by this, we refer to the interface $\hat{I}$ at the center of an RPO $\vec{B} : \vec{I} \to \hat{I}, B : \hat{I} \to L$) whose sort is large enough to sort the 'legs' $\vec{B}$ of the RPO. For pushout reflection, we have a pushout interface but need to prove that it is small enough to 'fit' the mediating arrow between the pushout and any other bound.

A.1 Useful lemmas

Lemma 11. *Let G be a bigraph of a partitioned s-category. For any root r of G, if $G(s) = r$ for some site of G then $sort(s) = sort(r)$.*

Proof. By condition K2, $sort(s) \subseteq sort(r)$. As the interface sorts are disjoint, $sort(s) = sort(r)$. □

Lemma 12. *Given two bigraphs $D : H \to J$ and $A : H \to I$ of a meet s-category $\mathcal{T}_{\mathrm{KB_h}}(\Sigma_{\mathrm{K}})$, let $B' : I^{\mathrm{u}} \to J^{\mathrm{u}}$ be a pure bigraph such that $D^{\mathrm{u}} = B' \circ A^{\mathrm{u}}$. Then there exists a bigraph $B : I \to J$ of $\mathcal{T}_{\mathrm{KB_h}}(\Sigma_{\mathrm{K}})$ such that $B^{\mathrm{u}} = B'$ and $D = B \circ A$.*



Proof. Any such B has the same *prnt* map as B'. We show that B is an arrow of $\mathcal{T}_{\mathrm{KB_h}}(\Sigma_{\mathrm{K}})$ *i.e.* that it obeys the kind sorting conditions and is fitting. In the following, v and v' are nodes, r is a site of B, s is a site, and p is a root or node of B.

K1 Let $p = B(v)$. Then $p = D(v)$ and since D is sorted, B obeys the sorting.

K2 Let $p = B(r)$. We must prove $sort(r) \subseteq sort(p)$.

As A is sorted, the sort of r is defined as

$$sort(r) = \mathcal{K}_r \uplus \big(\bigcup_{v <_A r} ctrl(v) \big) \cup \big(\bigcup_{s <_A r} sort(s) \big)$$

where by the definition of fitting s-category, $\mathcal{K}_r$ is the smallest possible set. As $D^{\mathrm{u}} = B' \circ A^{\mathrm{u}}$, we have $v <_A r$ implies $v <_D p$ and $s <_A r$ implies $s <_D p$. As D is sorted, the sort of p can be therefore be defined as

$$sort(p) = \mathcal{K}_p \uplus \big(\bigcup_{v <_A r} ctrl(v) \big) \cup \big(\bigcup_{s <_A r} sort(s) \big).$$

As $\mathcal{T}_{\mathrm{KB_h}}(\Sigma_{\mathrm{K}})$ is a meet s-category, the interface sort

$$sort(r) \cap sort(p) = \mathcal{K}' \uplus \big(\bigcup_{v <_A r} ctrl(v) \big) \cup \big(\bigcup_{s <_A r} sort(s) \big)$$

is defined. This sort can sort the root r in A and as $sort(r) \cap sort(p) \subseteq sort(r)$, and $sort(r)$ is the smallest sort to sort r, it must be that $sort(r) = sort(r) \cap sort(p)$. Thus, $sort(r) = sort(r) \cap sort(p) \subseteq sort(p)$.

K3 Trivially follows from $D^{\mathrm{u}} = B' \circ A^{\mathrm{u}}$.

We now prove B is fitting. Let t be a place in J. The smallest interface sort to sort t in D must contain the control of all children of t in D and the sorts of all sites which are children of t in D. The same sort is also the smallest interface sort to sort t in B. This can be seen by examining the composition $B \circ A$ and by the following argument – for any site s of B where $s <_B t$, all elements in $sort(s)$ are necessary to sort A as otherwise we can find a smaller sort $sort(s) \cap sort(t)$. □

The next four paragraphs are for those familiar with the recent work of Birkedal, Debois, and Hildebrandt on predicate sortings and their general RPO theorem for sorted reactive systems [2, Thm. 12]. While they worked in categories, we assume here that their results generalise to precategories. Their definitions capture and generalise notions in our work and so we shall adopt some of their terminology in the future.

The notions of inflations, cospans, and fitting pairs in $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ [9] correspond to (and are generalised by) verticals, nearly jointly opcartesian cospans, and jointly opcartesian cospans.

Lemma 12 seems to imply that the sorting functor $\mathcal{U}$ of meet s-categories is a *weak opfibration* (and further implies that all cospans in meet s-categories are *jointly opcartesian* with respect to the sorting functor). A similar lemma holds for the fitting s-category [9].

It would seem that any pair of kind bigraphs with common codomain such that the codomain has the least sort to sort both bigraphs is jointly opcartesian – this is the basis for the RPO construction in all the s-categories of Figure 2 which create RPOs. Further, all bigraphs in $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ are *nearly opcartesian* as they can decomposed into a fitting bigraph (which is opcartesian [9]) and an inflation (which is a *vertical*) and thus the sorting functor from this s-category is a *weak opfibration*. $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *reflects prefixes* (use the interface with the smallest sort) whenever no invisible controls are present. Fibres have pushouts in $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ (union the sorts to form the pushout interface), and we believe that a special case of the RPO construction would prove that these pushouts are pushouts in $\acute{}\mathrm{BIG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$. This all sits nicely with their general RPO theorem.

Not all fitting s-categories reflect prefixes (an interface which sorts the decomposition may not exist). Similarly, a kind s-category cannot reflect prefixes whenever invisible controls exist in the signature as these are not allowed in interface sorts (this may possibly be resolved by forgetting these in the sorting functor in cases where it is still possible to be faithful). These examples fall outside the general RPO theorem. We examine some of them in the next section.

A.2 RPO creation

Example 2 (fitting s-categories do not create RPOs in general)**.** *Let* $\acute{}\mathrm{FKB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *be a fitting s-category over*

$$\Sigma_{\mathrm{K}} = (\mathcal{K}, \mathsf{vis}^{\mathcal{K}}, \mathcal{P}(\mathcal{K}), \Phi)$$

where $\mathcal{K} = \{A, B, C, D\}$ *with all controls visible and of arity zero. Let* Φ *define* A*,* B*, and* C *as atomic and the sort of* D *as* $\{A, B, C\}$*.*

Consider the commuting diagram on the left of Figure 7 where the node of control A *is shared in* $A|B$ *and* $A|C$ *and we write* θ *for the interface* $\langle 1, \theta, \emptyset \rangle$ *and omit the sorts in the identity arrows*[4]*. The three bigraphs inside the outer square form an RPO of pure bigraphs. However, this is only an RPO in* $\acute{}\mathrm{FKB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *when the missing interface is defined as* $\{A, B, C\}$*. If this is not an interface sort then an RPO is not created.*

As an aside, we have not consider fitting unioned s-categories in this paper. Such s-categories do not create RPOs (see below) but may reflect pushouts. As unioned s-categories do create RPOs (Proposition 13) and we believe they weakly reflect pushouts, they seem preferable.

[4]We take some liberties with terminology here – they are only identities on places, not interfaces.

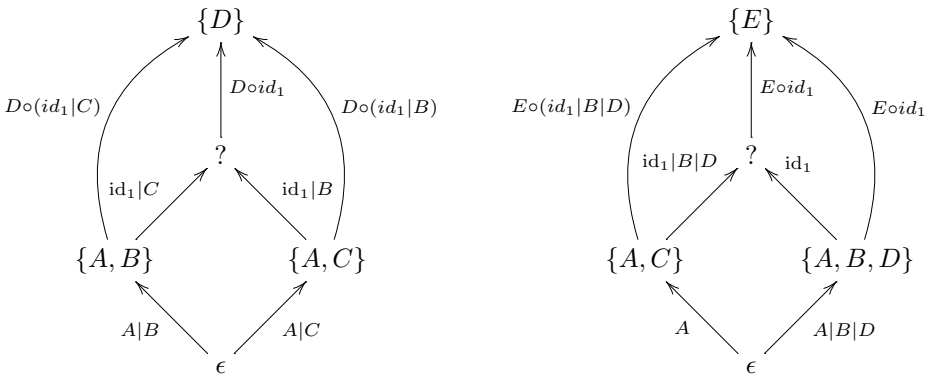


Figure 7: Diagrams for Example 2 (left) and Example 3 (right)

Example 3 (fitting unioned s-categories do not create RPOs in general)**.** *Let* $\text{Ƭ}\textsc{Kb}_{\text{h}}(\Sigma_{\text{K}})$ *be a fitting s-category over*

$$\Sigma_{\text{K}} = (\mathcal{K}, \mathsf{vis}^{\mathcal{K}}, \mathcal{P}(\mathcal{K}), \Phi)$$

where $\mathcal{K} = \{A, B, C, D, E\}$ *with all controls visible and of arity zero. Let* Φ *define* A*,* B*,* C*, and* D *as atomic and the sort of* E *as* $\{A, B, C, D\}$*.*

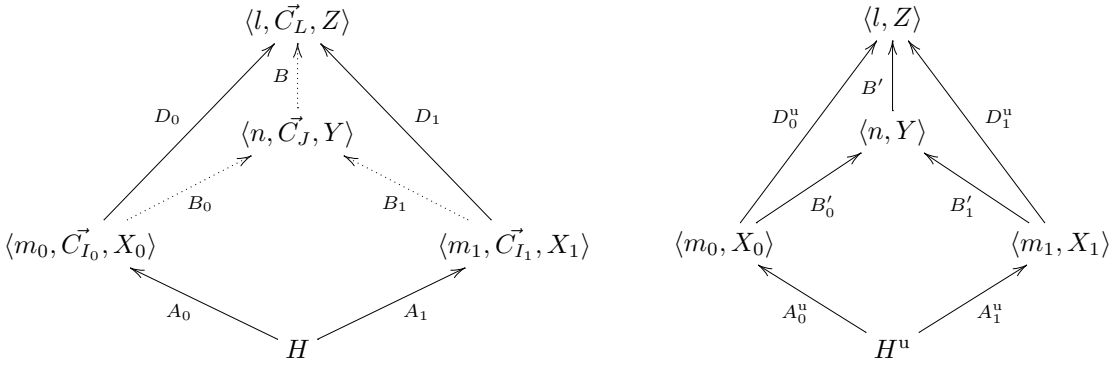
Consider the commuting diagram on the right of Figure 7 where the node of control A *is shared in* A *and* $A|B|D$*. The three bigraphs inside the outer square is an RPO of pure bigraphs. However, this is only an RPO in* $\text{Ƭ}\textsc{Kb}_{\text{h}}(\Sigma_{\text{K}})$ *when the missing interface is defined as* $\{A, B, C, D\}$*. But then the bigraph* $\text{id}_1 : \{A, B, D\} \to \{A, B, C, D\}$ *is not fitting.*

Construction 1 (building RPOs)**.** *Let* $\vec{A} : H \to \vec{I}$ *have a bound* $\vec{D} : \vec{I} \to L$ *in a subcategory* $\text{Ƭ}\mathbf{A}$ *of a kind s-category* $\text{Ƭ}\textsc{Big}_{\text{h}}(\Sigma_{\text{K}})$*. We define an RPO* $\vec{B} : \vec{I} \to \hat{I}, B : \hat{I} \to L$ *for this square as follows when:*

1. $\text{Ƭ}\mathbf{A}$ *is a downward closed and controlled s-category;*
2. $\text{Ƭ}\mathbf{A}$ *is a downward closed and unioned s-category;*
3. $\text{Ƭ}\mathbf{A}$ *is a unioned s-category (non-fitting);*
4. $\text{Ƭ}\mathbf{A}$ *is a partitioned s-category.*

The first steps for the different cases are the same. We start by building a pure RPO $(\vec{B}', B')$ for $\vec{A}^{\text{u}}$ to $\vec{D}^{\text{u}}$ [17, Construction 7.7]. From this we shall construct a kind sorted bound $(\vec{B}, B)$ for $\vec{A}$ to $\vec{D}$, such that $(\vec{B}, B)^{\text{u}} = (\vec{B}', B')$. Then in the next proposition we shall prove the universal

property of this relative bound. The situtation is depicted below.



For each case, we will define the sorts of places in the interface $\hat{I}$ and then prove that triple $(\vec{B}, B)$ is kind sorted and consists of bigraphs in the s-category we are working in. In order to define the sorts, we must revisit Jensen and Milner's construction and see how each place of $\hat{I}$ arises in the construction.

Let $\hat{r}$ be a place of $\hat{I}$. $\hat{r}$ is not barren in either B'_0 or B'_1[5]. Let r_i denote a site of B_i, $i \in \{0, 1\}$. The controls of nodes of B'_0 which are children of $\hat{r}$ and the nodes of B'_1 which are children of $\hat{r}$ are respectively defined as

$$\begin{aligned} V_{B'_0} &= \bigcup_{r_1 <_{B'_1} \hat{r}} \Big(\bigcup_{v <_{A_1} r_1} ctrl(v) \Big), \\ V_{B'_1} &= \bigcup_{r_0 <_{B'_0} \hat{r}} \Big(\bigcup_{v <_{A_0} r_0} ctrl(v) \Big). \end{aligned}$$

In order that $\hat{r}$ respects the kind rules for both B_0 and B_1, we must have

$$sort(\hat{r}) = \mathcal{K}_r \uplus \Big(\big(V_{B'_0} \cup V_{B'_1} \big) \cup \Big(\bigcup_{r_0 <_{B'_0} \hat{r}} sort(r_0) \Big) \cup \Big(\bigcup_{r_1 <_{B'_1} \hat{r}} sort(r_1) \Big) \Big)$$

for some smallest set $\mathcal{K}_r$ such that $sort(\hat{r})$ is an interface sort. However, as $B'_0 \circ A_0^{\mathrm{u}} = B'_1 \circ A_1^{\mathrm{u}}$, if $v_1 <_{B_0} \hat{r}$ then $v_1 <_{A_1} r_1$ for some $r_1 <_{B'_1} \hat{r}$. As A_0 and A_1 are sorted, we can then rewrite the equation above as

$$sort(\hat{r}) = \mathcal{K}_r \uplus \Big(\Big(\bigcup_{r_0 <_{B'_0} \hat{r}} sort(r_0) \Big) \cup \Big(\bigcup_{r_1 <_{B'_1} \hat{r}} sort(r_1) \Big) \Big). \tag{1}$$

Before we break the remainder of the construction over the cases, we state the following sub-proposition:

> If an interface which satisfies the equation (1) exists in the s-category this interface defines B_0 and B_1 as kind sorted bigraphs *i.e.* conditions **K1**-**K3** are satisfied. (P1)

By the definition of $sort(\hat{r})$, **K1** and **K2** are satisfied when p is a root. If p is a node, then the commutativity of the diagram above proves **K1**, **K2**, and **K3**. Similarly, commutativity of the diagram and the fact that no place of $\hat{I}$ is barren in B_0 or B_1 proves that **K1** and **K3** are satisfied in B.

[5]This is a familiar property in colimits of diagrams of sets and set-like categories.

We now explicitly define the sort of a place $\hat{r}$ of $\hat{I}$ for the separate cases. For the fitting s-categories, we must also show that this sort is the smallest sort satisfying (1) in order that B_0 and B_1 are arrows of that subcategory. We must also show that B satisfies **K2**.

1. For a downward closed and controlled s-category $\acute{}\mathbf{A}$, we define

$$sort(\hat{r}) \quad = \quad \Big(\bigcup_{r_0 <_{B'_0} \hat{r}} sort(r_0) \Big) \cup \Big(\bigcup_{r_1 <_{B'_1} \hat{r}} sort(r_1) \Big). \tag{2}$$

   We first prove that this is an interface sort.

   Let $\hat{r} <_{B'} p$ where p is a root or node. For any $r_i, r_i <_{B'_i} \hat{r}, i \in \{0,1\}$, we have $sort(r_i) \subseteq sort(p)$ as D_i is sorted. Therefore, $sort(\hat{r}) \subseteq sort(p)$. As $\acute{}\mathbf{A}$ is downwards closed and each control sort is an interface sort, $sort(\hat{r})$ is therefore an interface sort and does not break **K2** in B. We must prove that it is also the smallest sort for B_0 or B_1 individually *i.e.* that B_0 and B_1 are arrows of $\acute{}\mathbf{A}$.

   Say $K \in sort(r_1), r_1 <_{B_1} \hat{r}$. We will prove that either $v <_{B_0} \hat{r}, ctrl(v) = K$ or $K \in sort(r_0), r_0 <_{B_0} \hat{r}$. This suffices to prove that $sort(\hat{r})$ is the smallest interface sort for B_0. Similar reasoning holds for B_1. As $K \in sort(r_1)$ then either:

   (a) $v <_{A_1} r_1, ctrl(v) = K$.
   If v is a shared node then, by the RPO construction, there exists a site $r_0 \in m_0$ such that $v <_{A_0} r_0$ and $r_0 <_{B_0} \hat{r}$. Therefore, $K \in sort(r_0)$.
   If v is not shared then it is also a node of B_0 such that $v <_{B_0} \hat{r}$ and so it is necessary to sort B_0.

   (b) $s <_{A_1} r_1, K \in sort(s)$.
   This case is the same as for a shared node.

   (c) K is some other control unnecessary to sort A_1.
   As all subsets of an interface sort are interface sorts, this case never occurs in this fitting s-category.

   Therefore, the sort above is the smallest interface sort of the fitting s-category which sorts B_0 and B_1 and so these are arrows of the category. By Lemma 12, so is B.

2. For a downward closed and unioned s-category $\acute{}\mathbf{A}$, we define $sort(\hat{r})$ as in equation (2) above. This is an interface sort by definition and clearly the smallest which sorts both B_0 and B_1. The remainder of this case proceeds as in the last case.

3. For a unioned s-category $\acute{}\mathbf{A}$, we define $sort(\hat{r})$ as in equation (2) above. This is an interface sort by definition and clearly the smallest which sorts both B_0 and B_1 (although not necessarily the smallest sort for B_0 or B_1 individually). We must now prove that B satisfies **K2**.

   Let p be a node or root of B'. We must prove that if $\hat{r} <_B p$ then $sort(\hat{r}) \subseteq sort(p)$ for some $\hat{r} \in \hat{I}$. Let $K \in sort(\hat{r})$. Then $K \in sort(r_i)$ for some $r_i \in m_i$ where $r_i <_{B_i} \hat{r}$ and $i \in \{0,1\}$. But $r_i <_{D_i} p$ and D_i is sorted so $K \in sort(p)$. Therefore, any element of $sort(\hat{r})$ is an element of $sort(p)$ and B is sorted.

4. For a partitioned s-category, we begin by proving that all places in equation (1) have the same sort.

   From the pure RPO construction, $\hat{r}$ must be the parent of at least one site in either B_0 or B_1.

If $\hat{r}$ has no sites as children in B_1 then it must have exactly one site r_0 as a child in B_0 and so $sort(\hat{r}) = sort(r_0)$. This sorts B_0 as $\hat{r}$ has no other children in B_0. The children of $\hat{r}$ in B_1 are the children of r_0 in A_0 and so B_1 is also sorted.

Let $\hat{r}$ have a non-zero number of sites as children in both B_0 and B_1. If we examine the equivalence relation $\cong$ over these sites defined in the pure construction, we can see that the relation that the equivalence relation is based on relates sites which both parent a shared node. In a partitioned s-category, this implies they have the same sort. Therefore, for any pair $(r_0 \in m_0, r_1 \in m_1)$ such that $r_0 <_{B_0} \hat{r}, r_1 <_{B_1} \hat{r}$, $sort(r_0) = sort(r_1)$. Therefore, $sort(\hat{r}) = sort(r_i)$ where $r_i <_{B_i} \hat{r}$, $i \in \{0, 1\}$. As above, this sorts B_0 and B_1.

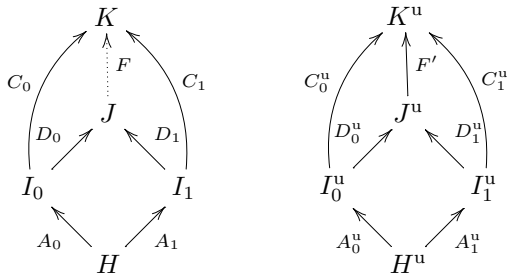
Finally, let $p >_{B'} \hat{r}$. Say $\hat{r}$ is the parent of a place $r_i \in m_i$ in B_i. Then $p >_{D_i} r_i$ and since D_i is sorted, $sort(\hat{r}) = sort(r_i) \subseteq p$. Therefore, B is sorted.

□

We will now prove that the constructions above indeed build RPOs. A crucial point is that the interface sort of the RPO interface is constructed to have the minimum available interface sort of the sub-category.

Proposition 13 (creation of RPOs)**.** *Whenever $\vec{D}$ bounds $\vec{A}$ in a subcategory $\acute{\mathbf{A}}$ of $\acute{\mathrm{B}}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$, then any RPO for $\mathcal{U}(\vec{A})$ relative to $\mathcal{U}(\vec{D})$ has a unique $\mathcal{U}$-preimage that is an RPO for $\vec{A}$ relative to $\vec{D}$ if:*

1. *$\acute{\mathbf{A}}$ is a downward closed and controlled s-category;*
2. *$\acute{\mathbf{A}}$ is a downward closed and unioned s-category;*
3. *$\acute{\mathbf{A}}$ is a unioned s-category;*
4. *$\acute{\mathbf{A}}$ is a partitioned s-category.*



Proof. The proof is similar for all cases. Start by using Construction 1 to build a canidate RPO $(\vec{B}, B)$. The interface defined by the construction is the same as for the construction of RPOs in the full s-category $\acute{\mathrm{B}}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ [9]. Therefore, given any other relative bound $(\vec{C}, C)$ in the subcategory, there is exactly one arrow F from the codomain of $\vec{B}$ to the codomain of $\vec{B}$ such that $F \circ B_i = C_i, i \in \{0, 1\}$. We need to prove that F is an arrow of the subcategory.

1. By Lemma 12.
2. By Lemma 12.
3. This is a full subcategory.
4. This follows as F is well-sorted.

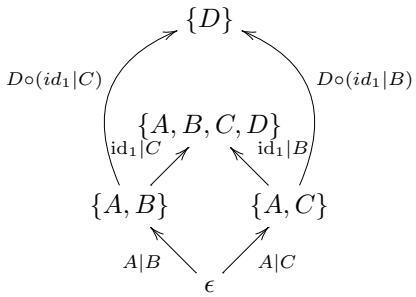
□

A.3 Pushout reflection

Example 4 (fitting s-categories do not reflect pushouts in general)**.** *Let* $\mathcal{T}\text{FKB}_\text{h}(\Sigma_\text{K})$ *be a fitting s-category over*

$$\Sigma_\text{K} = (\mathcal{K}, \mathsf{vis}^{\mathcal{K}}, \mathcal{P}(\mathcal{K}), \Phi)$$

where $\mathcal{K} = \{A, B, C, D\}$ *with all controls visible and of arity zero. Let* Φ *define* A*,* B*, and* C *as atomic and the sort of* D *as* $\{A, B, C\}$*. Define the set of interface sorts as* $\mathcal{P}(\mathcal{K}) \setminus \{A, B, C\}$*.*



Consider the commuting diagram above where the node of control A *is shared in* $A|B$ *and* $A|C$ *and we write* θ *for the interface* $\langle 1, \theta, \emptyset\rangle$ *and omit the sorts in the identity arrows. The lower square is a pushout of pure bigraphs. Therefore, there is a unique mediator* $D \circ id_1$ *from* $\{A, B, C, D\}^{\text{u}}$ *to* $\{D\}^{\text{u}}$ *from the inner bound to the outer bound. However, there is no such sorted mediator as the sort of* D *does not contain* D*.*

The problem in this example is that the minimum sort which contains the sort of some control D is a strict superset of $sort(D)$. This example led to the definition of meet s-categories. In that definition, it is important that θ' ranges over both interface sorts and control sorts as otherwise we can find a counterexample to pushout reflection.

A more general intuition may help. Consider a pushout for a pair of bigraphs (A_0, A_1) with common domain. If we examine the pushout construction [26] for pure place graphs, the pushout is the least identification on roots. By this, we mean that the roots of the outer interface of the pushout correspond to the equivalence classes of an equivalence relation on the roots of A_0 and A_1 (in fact, this is very similar to the construction of a pushout in the category of sets). It is the least identification in that it only puts two roots in the same equivalence class when necessary to form a bound. It also does not add any 'extra' roots.

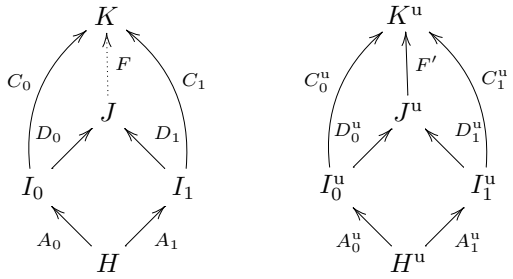
When all the equivalence classes contain exactly one root then pushout reflection should follow. For example, let an equivalence class contain only the root r of A_0. By the pushout construction [26], $A_0 = \pi(P \otimes G)$, $B_0 = \pi'(\text{id}_{sort(r)} \otimes G')$, and $B_1 = \pi''(P \otimes G'')$ for some prime P with outer sort $sort(r)$, permutations π, π', π'', and bigraphs G,G', and G''. Pushout reflection is then easily proved – for any other bound (C_0, C_1) for (A_0, A_1), the unique mediator from the pushout to (C_0, C_1) is a reflection of the corresponding pure mediator (see the following proofs). We must prove that this mediator is well-sorted and an arrow of the subcategory. The only hard condition to satisfy is **K2** and the facts that $C_0 \circ A_0$ is well-sorted and that each equivalence class has one member is enough to guarantee **K2**.

A general proof fails, as the example above shows, as roots may be identified in the pushout. According to our definition of fitting s-category and the pure pushout construction, if two roots with sorts θ and θ' are identified, then the place of the pushout interface which parents these in the pushout has the sort $\mathcal{K}_P \uplus (\theta \cup \theta')$ for some smallest set $\mathcal{K}_P$. In a general proof argument, where the set of interface sorts is not known, we do not know what the set $\mathcal{K}_P$ contains and our argument falls over. In the example above, the sorts $\{A, B\}$ and $\{A, C\}$ were merged into a sort $\{A, B, C, D\}$. The inclusion of $\{D\} = \mathcal{K}_P$ is what broke the example.

Unioned s-categories do not reflect pushouts either. In the s-category of Example 4, $\{A, B, C\}$ and $\{A, B, C, D\}$ are both interface sorts and both can sort the bottom square of the diagram. However, these two bounds may only both be pushouts if they are isomorphic which is not the case as they have different sorts. Therefore, pushouts are not reflected in general in unioned s-categories. The problem here is that we do not force outer interfaces of bigraphs to be minimal as we do in fitting s-categories. However, unioned s-categories may *weakly* reflect pushouts

Proposition 14 (reflection of pushouts)**.** *Whenever $\vec{D}$ bounds $\vec{A}$ in $\mathcal{F}\mathrm{KB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ and $\mathcal{U}(\vec{D})$ is a pushout for $\mathcal{U}(\vec{A})$, then $\vec{D}$ is a pushout for $\vec{A}$ if:*

1. *$\mathcal{F}\mathrm{KB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ is a meet s-category;*
2. *$\mathcal{F}\mathrm{KB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ is a partitioned s-category.*



Proof. To prove that $\vec{D}$ is a pushout for $\vec{A}$, we first show that for any other bound $\vec{C}$ for $\vec{A}$, there exists a sorted mediator F such that $F \circ D_i = C_i$ and then prove the uniqueness of F. The situation is depicted in the diagram above.

Assume that $\vec{C}$ is another bound for $\vec{A}$. Let F' be the unique (pure) mediating arrow from $(D_0^{\mathrm{u}}, D_1^{\mathrm{u}})$ to $(C_0^{\mathrm{u}}, C_1^{\mathrm{u}})$. Let F have the same *prnt* map as F'.

First, we must show that F is an arrow of $\mathcal{F}\mathrm{KB}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ *i.e.* that it is sorted.

1. The proof follows from Lemma 12.
2. We break the proof over the kind sorting conditions. In the following, v and v' are nodes, r is a site of F, and p is a root or node of F.

   K1 Let $p = F(v)$. Then $p = C_0(v)$ and since C_0 is sorted, F obeys the sorting.

   K2 Let $p = F(r)$. We must prove $sort(r) \subseteq sort(p)$.

   To prove this, we will examine the construction of pushouts in pure place graphs [26]. The root r must be a parent of a site $s \in I_0$ or a node v in D_0.

   Let $s <_{D_0} r$. By Lemma 11, $sort(r) = sort(s)$. As $C_0^{\mathrm{u}} = F' \circ D_0^{\mathrm{u}}$, $s <_{C_0} p$. Therefore, $sort(r) = sort(s) \subseteq sort(p)$.

   Let $v <_{D_0} r$. Then, by the construction of pushouts, $v <_{A_1} s' \in I_1$ and $s' <_{D_1} r$. By Lemma 11, $sort(r) = sort(s')$. As $C_1^{\mathrm{u}} = F' \circ D_1^{\mathrm{u}}$, $s' <_{C_1} p$. Therefore, $sort(r) = sort(s') \subseteq sort(p)$.

   K3 Follows from $C_0^{\mathrm{u}} = F^{\mathrm{u}} \circ D_0^{\mathrm{u}}$.

F is a sorted mediator between the two bounds $\vec{D}$ and $\vec{C}$. Lemma 12 and the definition of partioned s-categories imply that F is an arrow in the s-category. The uniqueness of F follows from the fact that $\mathcal{U}$ is faithful and that F' is a unique mediator. □

B Two generalisations of kind sortings

In this section, we present two generalisations of kind sortings. We defer proofs of RPO creation and weak pushout reflection for the first until later (in case we further generalise) but we are confident that the existing proofs for kind sorting need only be slightly modified.

B.1 Kind sorting with capacities

The first generalisation allows the number of children of a node to be bounded. This allows us to properly model multi-nodes which in turn allows us to model ordered tree structures in bigraphs. An immediate application of this sorting would be to model XML data whose document order is relevant. It also allows us to better model the λ-calculus as we will suggest below.

The idea behind the sorting is as follows. It allows us to specify the number of K'-nodes that a K node will contain in a bigraph, for each K' in some signature $\mathcal{K}$. We can specify a lower bound and/or an upper bound, and the upper bound may be "arbitarily many." We also specify a *separate* pair of bounds per control. The minimum capacity of K is the number of arbitrary nodes that a K-node *must at least* contain. The maximum capacity of K is the number of arbitrary nodes that a K-node *may at most* contain. Again, the maximum capacity is allowed to be "arbitarily many."

This idea is uncomplicated but we need to ensure that the sorting is preserved by composition. We therefore introduce some machinery to ease the definition of, and hopefully the reasoning about, the sorting.

Definition 17 ($\mathbb{N}^{\odot}$)**.** *We define the set $\mathbb{N}^{\odot} = \mathbb{N} \cup \{\odot\}$. We extend the partial order $\leq$ by defining $n \leq \odot, n \in \mathbb{N}^{\odot}$ and extend the addition operation with $n + \odot = \odot, n \in \mathbb{N}^{\odot}$.*

The symbol $\odot$ represents "non-zero." This will be used to specify that a node of a bigraph may contain finitely many other nodes.

Notation (projections)**.** *For every product $S_1 \times \cdots \times S_n$ of n sets there are n canonical projection maps $\pi_i : S_1 \times \cdots \times S_n \to S_i, 1 \leq i \leq n$. We extend this notation to any subset $T \subseteq S_1 \times \cdots \times S_n$ of an n-ary product in the obvious manner. We sometimes write $\pi_i(P)$ meaning instead the image of the projection.*

In other words, given a tuple $c \in T \subseteq S_1 \times \cdots \times S_n$, $\pi_i(c)$ denotes the i^{th} element of c.

Notation. *For the remainder of this section we use the following notation.*

1. *We write $(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$ to denote the largest subset of $\mathbb{N} \times \mathbb{N}^{\odot} \times S$ where $\pi_1(c) \leq \pi_2(c)$ for all elements.*
2. *We write $\mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$ to denote the largest subset of $\mathcal{P}((\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq})$ where for all elements T and $c \in T$, $\pi_3 : T \to S$ is a bijection.*
3. *We write $(\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S))^{\leq}$ for the largest subset of $(\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq})^{\leq}$, on which the following condition also holds on elements T:*

$$\sum_{c \in \pi_3(T)} (\pi_1(c)) \;\leq\; \pi_1(T) \;\leq\; \pi_2(T) \;\leq \sum_{c \in \pi_3(T)} (\pi_2(c)).$$

These definitions help us to reason about lower and upper bounds on capacities. The first definition states that we will be considering triples where the first element is a lower bound and the second an upper bound. Informally, every time you see a triple, the first element (an integer) is going to be smaller than the second (an integer or $\odot$).

The injectivity of π_3 in the second definition ensures that the definition below is unambiguous – that the sort of a control may only contain one triple (m, n, K) for each control K. This triple states that a control must contain at least m nodes of K and may at most contain n nodes, where $m \leq n$. The surjectivity of π_3 is unnecessary but makes the following theory simpler.

The final definition is more involved. Informally, the first inequality will state that the number of controls a node must at least contain is (possibly) greater than the sum, ranging over $\mathcal{K}$, of the number of controls of type $K \in \mathcal{K}$ it must at least contain. Similarly, the last inequality will state that the number of controls a node may contain is (possibly) less than the sum, ranging over $\mathcal{K}$, of the number of controls of type $K \in \mathcal{K}$ it can contain. These inequalities relax the idea of lower and upper bounds somewhat, where the "space between" both sides may be filled in arbitrarily. The middle inequality is redundant (due to the first definition) but we include it to help the presentation.

Definition 18 (kind signature with capacities)**.** *A* kind signature with capacities $\{\mathcal{K}, arity, actv, vsbl, cpc\}$ *is composed of a set* $\mathcal{K}$ *of* controls *and four maps:*

$$\begin{aligned} arity &: \mathcal{K} \to \mathbb{N} \\ actv &: \mathcal{K} \to \{\mathsf{passive}, \mathsf{active}\} \\ vsbl &: \mathcal{K} \to \{\mathsf{vis}, \mathsf{inv}\} \\ cpc &: \mathcal{K} \to (\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{K}))^{\leq} \end{aligned}$$

We define $\mathsf{min}_{cpc}(K) = \pi_1(cpc(K))$, $\mathsf{max}_{cpc}(K) = \pi_2(cpc(K))$, *and* $kind(K) = \pi_3(cpc(K))$.

We call $cpc(K) = sort(K)$ the *capacity* or *sort* of K and $kind(cpc(K))$ the *kind* of K[6]. The first two elements of the sort specifies a *minimum capacity* $\mathsf{min}_{cpc}(K)$ and a *maximum capacity* $\mathsf{max}_{cpc}(K)$ for each control K. These respectively represent how many nodes of any control a K-node must at least and may at most contain. The third element is a subset of $\mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{K}) \leq$ which associates two positive quantities with each control K' that K may contain. These quantities respectively represent how many K'-nodes a K-node must at least contain and may at most contain. We ignore listing elements of $\mathcal{K}$ when they cannot be contained.

Convention. *When we are discussing the kind of a control or interface sort (see below), we typically ignore triples* $(0, 0, K)$ *for arbitrary* K.

Again, there is a difference between the minimum (resp. maximum) capacity and the sum of the controls a K-node may at least (resp. must at most) contain. For example, we may wish to specify that a K-node may contain exactly one node, either a K'-node or a K''-node. We can then define $cpc(K) = (1, 1, \{(0, 1, K'), (0, 1, K'')\})$ which satisfies this specification.

A control K is said to be *atomic* if its kind is $\{0\} \times \{0\} \times \mathcal{K}$ (and thus its minimum and maximum capacities are zero), otherwise *non-atomic*. Atomic controls may not be active. The function $vsbl$ partitions $\mathcal{K}$ into two sets $\mathcal{K}_{\mathsf{vis}}$ and $\mathcal{K}_{\mathsf{inv}}$ of *visible* and *invisible* controls.

The definition generalises kind signatures. Kind signatures are now a special case where $cpc : \mathcal{K} \to \{0\} \times \{\odot\} \times \mathcal{P}(\{0\} \times \{\odot\} \times \mathcal{K})$.

We can now define multi-nodes as elements of a kind signature with capacities.

Definition 19 (m-node of a kind sorting)**.** *In a kind signature with capacities* $\mathcal{K}$*, if a visible control* K *has the sort* $(m, m, (\{1, 1\} \times \{K_1, \ldots \mathcal{K}_m\})$*, where each* $K_i, 1 \leq i \leq m$ *is invisible, then we say that* K *is an* m*-node of* $\mathcal{K}$.

[6]Defining $cpc(K) = sort(K)$ makes the sorting condition easier to write. However, the name *cpc* is not considered redundant as it allows us to distinguish different sorts in *e.g* a paired sorting (see Appendix B.2).

Definition 20 (join, $\curlyvee$)**.** *Let $T_0, T_1 \in \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$ for some S. We define the* join *$T_0 \curlyvee T_1$ as*

$$T_0 \curlyvee T_1 \quad = \quad \{(m_0 + m_1, n_0 + n_1, K) \mid (m_0, n_0, K) \in T_0 \text{ and } (m_1, n_1, K) \in T_1\}$$

The join operation is part sum, part union and is commutative and associative. It is closely related to the the join operation of unordered multisets (bags) but as the symbol $\uplus$ has another meaning in the bigraph literature, we use $\curlyvee$.

Definition 21 (subkind, superkind)**.** *The subkind relation $\sqsubseteq$ on the set $\mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$ is defined as the smallest equivalence relation containing the relation*

$$T_1 \sim T_2 \quad \text{if for } (m_1, n_1, s) \in T_1, (m_2, n_2, s) \in T_2 \text{ we have } m_1 \geq m_2 \text{ and } n_1 \leq n_2$$

where $T_1, T_2 \in \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$. The superkind relation $\sqsupseteq$ is defined as the inverse of $\sqsubseteq$.

Definition 22 (interface sorts)**.** *To every kind signature with capacities $\mathcal{K}$, we associated a set of* interface sorts *defined as*

$$\Theta_{\mathcal{K}} = (\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{K}_{\mathsf{vis}}))^{\leq}.$$

If a place r is assigned a sort $\theta \in \Theta_{\mathcal{K}}$, we define $\mathsf{min}_{cpc}(r) = \pi_1(\theta)$, $\mathsf{max}_{cpc}(r) = \pi_2(\theta)$, and $kind(r) = \pi_3((\theta))$.

Lemma 15. *If $R_0 \sqsubseteq T_0$ and $R_1 \sqsubseteq T_1$ then $R_0 \curlyvee R_1 \sqsubseteq T_0 \curlyvee T_1$.*

Proof. Let i range over $\{0, 1\}$ and define $\bar{\imath} = 1 - i$. All sets are elements of $\mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times S)^{\leq}$ for some S.

Let $(h, k, s) \in R_0 \curlyvee R_1$ and $(m, n, s) \in T_0 \curlyvee T_1$. We need to prove that $h \geq m$ and $k \leq n$. Let $(h_i, k_i, s) \in R_i$ with $h = h_0 + h_1, k = k_0 + k_1$. As $R_i \sqsubseteq T_i$, $(m_i, n_i, s) \in T_i$ with $m = m_0 + m_1, n = n_0 + n_1$, $h_i \geq m_i$, and $k_i \leq n_i$. Therefore, $h = h_0 + h_1 \geq m_0 + m_1 = m$ and $k = k_0 + k_1 \leq n_0 + n_1 = n$. □

The subkind relation captures our intended notion of smaller kind. If $kind(w) \sqsubseteq kind(p)$ then this means that p is required to contain less K-nodes and can contain more K-nodes than w.

Definition 23 (kind sorting with capacities)**.** *A place-sorting $\Sigma_{\mathrm{K}} = (\mathcal{K}, \Theta, \Phi)$ over a kind signature with capacities $\mathcal{K}$ is a* kind sorting with capacities *if $\Theta = \Theta_{\mathcal{K}}$ and Φ requires for all bigraphs G that:*

$$\textbf{KC1} \qquad \mathsf{min}_{cpc}(p) \;\leq\; \sum_{K \in \mathcal{K}} p_{K,G} + \sum_{s <_G p} \mathsf{min}_{cpc}(s)$$

$$\textbf{KC2} \qquad \mathsf{max}_{cpc}(p) \;\geq\; \sum_{K \in \mathcal{K}} p_{K,G} + \sum_{s <_G p} \mathsf{max}_{cpc}(s)$$

$$\textbf{KC3} \qquad kind(p) \;\sqsupseteq\; \bigcup_{K \in \mathcal{K}} (p_{K,G}, p_{K,G}, K) \;\curlyvee\; \bigvee_{s <_G p} kind(s)$$

where p is a root or node, $p_{K,G}$ is the number of nodes v with control K where $v <_G p$, and s is a site.

In order, the conditions ensure that each root or node:

1. reaches at least the minimum capacity;
2. does not exceed the maximum capacity;
3. contains only K-nodes it is supposed to and in an allowed quantity.

This sorting is satisfied by identities and preserved by tensor product.

Proposition 16. *Composition preserves kind sorting with capacities.*

Proof. Let $D = B \circ A$ with A and B sorted. We will prove that D is sorted. Let $p_{K,G}$ denote the number of nodes of control K under a root or node p in a bigraph G. Let r denote sites of B and s denote sites of A. The definition of composition yields the two equations below.

$$\sum_{K\in\mathcal{K}} p_{K,B\circ A} = \sum_{K\in\mathcal{K}} \big(p_{K,B} + \sum_{r<_B p} r_{K,A}\big) \tag{1}$$

$$\{s \mid s <_{B\circ A} p\} = \{s \mid s <_A r, r <_B p\} \tag{2}$$

KC1

$$\begin{aligned}
&\langle A \text{ is sorted}\rangle\\
\Rightarrow\quad & \forall r <_B p.\big(\mathsf{min}_{cpc}(r) \le \sum_{K\in\mathcal{K}} r_{K,A} + \sum_{s<_A r} \mathsf{min}_{cpc}(s)\big)\\
\Rightarrow\quad & \sum_{r<_B p} \mathsf{min}_{cpc}(r) \le \sum_{r<_B p} \big(\sum_{K\in\mathcal{K}} r_{K,A} + \sum_{s<_A r} \mathsf{min}_{cpc}(s)\big)\\
\Rightarrow\quad & \sum_{r<_B p} \mathsf{min}_{cpc}(r) \le \sum_{K\in\mathcal{K}} \sum_{r<_B p} r_{K,A} + \sum_{r<_B p} \sum_{s<_A r} \mathsf{min}_{cpc}(s)\\
\Rightarrow\quad & \sum_{K\in\mathcal{K}} p_{K,B} + \sum_{r<_B p} \mathsf{min}_{cpc}(r) \le \sum_{K\in\mathcal{K}} \big(p_{K,B} + \sum_{r<_B p} r_{K,A}\big) + \sum_{r<_B p} \sum_{s<_A r} \mathsf{min}_{cpc}(s)\\
&\langle \text{equations (1) and (2)}\rangle\\
\Rightarrow\quad & \sum_{K\in\mathcal{K}} p_{K,B} + \sum_{r<_B p} \mathsf{min}_{cpc}(r) \le \textstyle\sum_{K\in\mathcal{K}} p_{K,B\circ A} + \displaystyle\sum_{s<_{B\circ A} p} \mathsf{min}_{cpc}(s)\\
&\langle B \text{ is sorted}\rangle\\
\Rightarrow\quad & \mathsf{min}_{cpc}(p) \le \sum_{K\in\mathcal{K}} p_{K,B} + \sum_{r<_B p} \mathsf{min}_{cpc}(r) \le \textstyle\sum_{K\in\mathcal{K}} p_{K,B\circ A} + \displaystyle\sum_{s<_{B\circ A} p} \mathsf{min}_{cpc}(s)
\end{aligned}$$

KC2 As above, changing min_{cpc} to max_{cpc} and $\le$ to $\ge$.

KC3 Let $p_{\mathcal{K},G}$ denote $\bigcup_{K\in\mathcal{K}} (p_{K,G}, p_{K,G}, K)$.

$$\begin{aligned}
&\langle A \text{ is sorted}\rangle\\
\Rightarrow\quad & \forall r <_B p.\big(kind(r) \sqsupseteq r_{\mathcal{K},A} \curlyvee \curlyvee_{s<_A r} kind(s)\big)\\
&\langle \text{repeated application of Lemma 15}\rangle\\
\Rightarrow\quad & \curlyvee_{r<_B p} kind(r) \sqsupseteq \curlyvee_{r<_B p} \big(r_{\mathcal{K},A} \curlyvee \curlyvee_{s<_A r} kind(s)\big)\\
\Rightarrow\quad & \curlyvee_{r<_B p} kind(r) \sqsupseteq \curlyvee_{r<_B p} r_{\mathcal{K},A} \curlyvee \curlyvee_{r<_B p} \curlyvee_{s<_A r} kind(s)\\
&\langle \text{Lemma 15}\rangle\\
\Rightarrow\quad & p_{\mathcal{K},B} \curlyvee \curlyvee_{r<_B p} kind(r) \sqsupseteq p_{\mathcal{K},B} \curlyvee \curlyvee_{r<_B p} r_{\mathcal{K},A} \curlyvee \curlyvee_{r<_B p} \curlyvee_{s<_A r} kind(s)\\
&\langle \text{equations (1) and (2)}\rangle\\
\Rightarrow\quad & p_{\mathcal{K},B} \curlyvee \curlyvee_{r<_B p} kind(r) \sqsupseteq p_{\mathcal{K},B\circ A} \curlyvee \curlyvee_{s<_{B\circ A} p} kind(s)\\
&\langle B \text{ is sorted}\rangle\\
\Rightarrow\quad & kind(p) \sqsupseteq p_{\mathcal{K},B} \curlyvee \curlyvee_{r<_B p} kind(r) \sqsupseteq p_{\mathcal{K},B\circ A} \curlyvee \curlyvee_{s<_{B\circ A} p} kind(s)
\end{aligned}$$

□

The sorting functor which forgets sorts is surjective on objects and faithful as usual. Kind sorting with capacities is another example of place-sorting and forms s-categories of bigraphs.

Definition 24 (kind sorted s-category/Brs with capacities)**.** *The kind sorted bigraphs with capacities (or Σ_{K}-sorted bigraphs) form a subcategory of $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\mathcal{K}, \Phi)$ denoted by $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$. We call $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}})$ a* kind sorted s-category with capacities*. If $\mathcal{R}$ is a set of Σ_{K}-sorted reaction rules then $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_{\mathrm{K}}, \mathcal{R})$ is a Σ_{K}-*sorted *(or* kind sorted *Brs with capacities.*

We will not present a proof of RPO creation here. We believe that RPO creation and weak pushout reflection hold but not (strong) pushout reflection.

The trick as usual is to identify which bigraphs have the least outer interface sort, or to use the proper terminology [2], to identify the opcartesian arrows of $\mathcal{B}\mathrm{IG}(\Sigma_{\mathrm{K}})$. This question is answered by Definition 23 and the definition of subsort. The smallest interface sort to sort a bigraph in the s-category will have, on each root: i) the maximum minimum capacity; ii) the minimum maximum capacity; iii) the smallest kind *i.e.* will only contain triples (m, n, K) where K is necessary, m is maximum, and n is minimum. In short, the inequalities in Definition 23 will be equalities for this sort.

To prove RPO creation, we require that each cospan $H_0 : I_0 \to J, H_1 : I_1 \to J$ is nearly jointly opcartesian meaning that there exists a minimal cospan $G_0 : I_0 \to J', G_1 : I_1 \to J'$ and a mediating arrow $F : J' \to J$ such that $F \circ G_i = H_i, i \in 0, 1$. We believe that this is the case for $\mathcal{B}\mathrm{IG}(\Sigma_{\mathrm{K}})$ and that it should not be hard to prove – the outer interface sort of a jointly opcartesian cospan should have the same properties as the outer interface sort of opcartesian arrows, except that we now take minimum and maximum numbers which properly sort both bigraphs. The numbers may not be maximum/minimum for both bigraphs, but will be for the pair.

Subcategories We can consider subcategories of sorted s-categories based on those in Definition 10. The notion of least sort of a root would seem to fall out of Definition 23 by replacing the inequalities in the conditions with equalities, leading to a definition of the standard fitting s-category. However, other odd subcategories may exist which create RPOs (assuming the full s-category does). For example, the subcategory where each number in the components of an interface sort θ is a multiple of some integer. Such combinatorics introduce a whole range of weird subcategories. However, we do not take a stance as to whether or not they could be useful for modelling.

Applications We have mentioned modelling XML data. Another application is to better model various calculi in bigraphs. For example, we could specify that all lam, def, $\mathsf{1}$, $\mathsf{2}$, and D controls in $\Lambda\mathrm{BIG}^{\tau}$ have a minimum and maximum capacity of 1 and that all nodes of control app (resp. sub) can contain exactly one $\mathsf{1}$ node and one $\mathsf{2}$ (resp. D) node. This removes many "junk" bigraphs. Most or all of the remaining junk can be eliminated by reducing the set of interface sorts, forming a subcategory. However, care must be taken here to preserve the transition theory. Constraining interface sorts to have a minimum and maximum capacity of one should yield a close match between Λ_{sub} terms and prime bigraphs.

Bundgaard and Hildebrandt posed the question of whether the sorting used for the bigraphical presentation of the Homer calculus with explicit substitutions could be expressed in any other sorting scheme [6, 7]. It seems that our extension of kind sorting is able to express the *place graph* restrictions of their sorting, using a sorting represented in Figure 8. In the figure, the labels on edges specify that nodes of the target control will contain exactly one node of the source control in each bigraph.

Jensen and Milner describe a method of turning an s-category of bigraphs into an s-category of hard bigraphs by adding a special atomic control, a place node $\triangle$, as a child of any barren

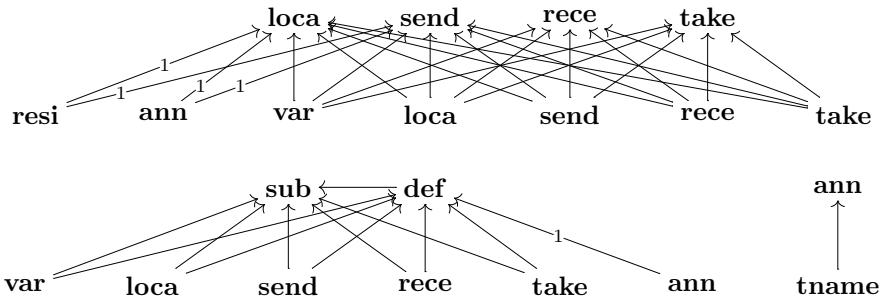


Figure 8: Representation of the place graph sorting of Homerσ

roots or barren non-atomic node [17]. Kind sorting with capacities could also be used as a means of specifying a certain kind of hard bigraph where each non-atomic node and interface sort must contain exactly one place node. This yields a different s-category than Jensen and Milner's as $\triangle$s are added to all roots and non-atomic nodes. It would be interesting to see whether or not this s-category could be used for the same technical purposes as theirs (a proof of engaged congruence in soft abstract Brss).

Further generalisation We could also consider treating sites like special types of controls and adding them to the signature. For example, the *cpc* function would be defined as

$$cpc : \mathcal{K} \to (\mathbb{N} \times \mathbb{N}^{\odot} \times \mathcal{P}(\mathbb{N} \times \mathbb{N}^{\odot} \times (\mathcal{K} \uplus \{site\})))^{\leq}$$

allowing us to specify how many sites a control can contain. The sorting extends similarly to interface sorts. It is not clear whether this would be useful.

B.2 Paired sorting

The second generalisation is a combination of kind sorting with the rigid control-sorting of Birkedal et al. [4], an instance of which has been proposed as a means of modelling context-aware systems. We first introduce a simple notion of combining sortings based on the definitions of link sorting and place sorting. The combination pairs two sortings without mixing any of their properties.

Definition 25. *We define the product* $\vec{\theta}_1 \times \vec{\theta}_2$ *of two vectors* $\vec{\theta}_1$ *and* $\vec{\theta}_2$ *of equal length* n *as the pointwise product* $\{\theta_1^1 \times \theta_2^1, \ldots, \theta_1^n \times \theta_2^n\}$.

Definition 26 (paired sorting)**.** *Given two sortings* $\Sigma_1 = (\mathcal{K}, \Theta_1, \Phi_1)$ *and* $\Sigma_2 = (\mathcal{K}, \Theta_2, \Phi_2)$ *over compatible signatures*[7]*, the* paired sorting *is the sorting* $\Sigma_1 \times \Sigma_2 = (\mathcal{K}, \Theta_1 \times \Theta_2, \Phi_1 \wedge \Phi_2)$ *on* $\Theta_1 \times \Theta_2$*-sorted bigraphs over* $\mathcal{K}$.

The condition $(\Phi_1 \wedge \Phi_2)(G)$ *on* $G : \langle m, X, \vec{\theta}_1 \times \vec{\theta}_2 \rangle \to \langle n, Y, \vec{\theta'}_1 \times \vec{\theta'}_2 \rangle \in \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_1 \times \Sigma_2)$ *is true exactly when for any pair of arrows*

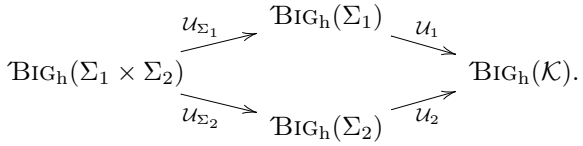
$$G_1 : \langle m, X, \vec{\theta}_1 \rangle \to \langle n, Y, \vec{\theta'}_1 \rangle \in \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_1),$$

$$G_2 : \langle m, X, \vec{\theta}_2 \rangle \to \langle n, Y, \vec{\theta'}_2 \rangle \in \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_2)$$

with the same underlying bigraph, $\Phi_1(G_1) \wedge \Phi_2(G_2)$.

When we are discussing paired sortings, we will let i range over $\{1, 2\}$. If $sort(r) = (\theta_1, \theta_2)$ for some place r (in a paired sorting of place-sortings) then we define $sort_1(r) = \theta_1$ and $sort_2(r) = \theta_2$.

Associated with a paired sorting are two forgetful functors $\mathcal{U}_{\Sigma_1}$ and $\mathcal{U}_{\Sigma_2}$ which forget a sort. These functors are surjective on objects and faithful and we have the following commuting diagram:



Lemma 17. $(\Phi_1 \wedge \Phi_2)(G) \equiv \Phi_1(\mathcal{U}_{\Sigma_2}(G)) \wedge \Phi_2(\mathcal{U}_{\Sigma_2}(G))$.

Proof. By Definition 26. □

Proposition 18. *If the functors* $\mathcal{U}_i : \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_i) \to \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\mathcal{K})$ *create RPOs then so does* $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$.

Proof. To save space, we will use the symbols □ and △ to respectively represent a commuting square and a relative bound/RPO for the square in $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_1 \times \Sigma_2)$ and talk about the image of these shapes under different functors. The situation is depicted in Figure 9.

The square $\mathcal{U}_i(\mathcal{U}_{\Sigma_i}(\square)) = \square^{\mathrm{u}}$ has a pure RPO $\triangle^{\mathrm{u}}$. Therefore, there are sorted RPOs $\triangle_1$ and $\triangle_2$ such that $\mathcal{U}_1(\triangle_1) = \mathcal{U}_2(\triangle_2) = \triangle^{\mathrm{u}}$ where the respective interface sorts of the RPO interface are $\vec{\theta}_1$ and $\vec{\theta}_2$. Define the candidate RPO $\triangle$ for □ as the underlying RPO $\triangle^{\mathrm{u}}$ with the interface sort $\vec{\theta}_1 \times \vec{\theta}_2$.

We first prove this triple is sorted. We have $\mathcal{U}_{\Sigma_i}(\triangle) = \triangle_i$. By Lemma 17, $\triangle$ is a triple of arrows in $\mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_1 \times \Sigma_2)$ and is a relative bound for □.

Now, let $\triangle'$ be any other relative bound for □. As $\triangle_i$ is an RPO, there are unique mediators F_i from $\triangle_i$ to $\mathcal{U}_{\Sigma_i}(\triangle')$ where $\Phi_i(F_i)$ such that $\mathcal{U}_1(F_1) = \mathcal{U}_2(F_2)$. By Definition 26, there is a unique mediator F from $\triangle$ to $\triangle'$. □

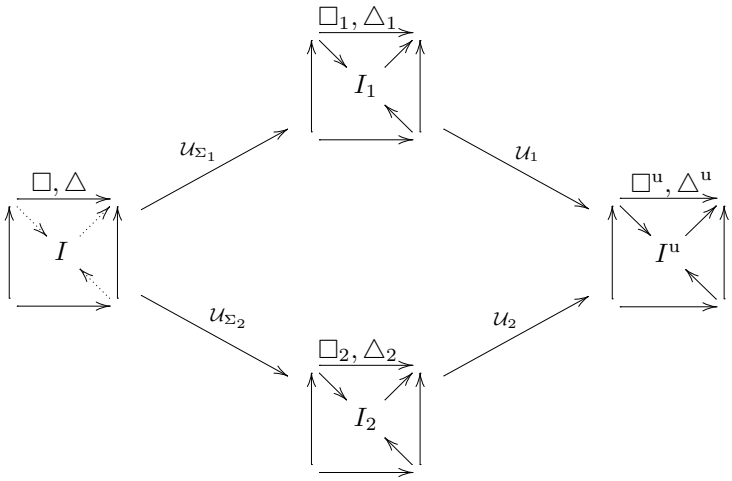


Figure 9: Constructing a paired sorted RPO

The definition of paired sorting and the proof of RPO creation are easily generalised for combinations of multiple sortings. This is perhaps unsurprising since the paired definition treats the two sortings somewhat separately and so the two different RPOs can be combined easily into one paired RPO, all with the same underlying bigraph.

Proposition 19. *If the functors* $\mathcal{U}_i : \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\Sigma_i) \to \mathcal{B}\mathrm{IG}_{\mathrm{h}}(\mathcal{K})$ *(weakly) reflect pushouts then so does* $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$.

Proof. We will prove that $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$ strongly reflects pushouts. For a proof of weak reflection, replace 'bound' with 'IPO' in the argument below, noting that Proposition 18 implies that the $\mathcal{U}_{\Sigma_i}$-images of an IPO are IPOs.

Figure 10 depicts the situation as in the last proposition. This time, $\triangle$ represents a bound (the lower left triangles of arrows in the diagram) which we will prove is a pushout and $\square$ represents some arbitrary bound (the squares of arrows in the diagram). Proving $\triangle$ is a pushout means proving that there is a unique mediator $F : I \to J$.

Let the bound $\mathcal{U}_i(\mathcal{U}_{\Sigma_i}(\triangle)) = \triangle^{\mathrm{u}}$ be a pure pushout. As $\mathcal{U}_i$ reflects pushouts, the bounds $\triangle_1$ and $\triangle_2$ are pushouts. There are then unique mediators $F_i : I_i \to J_i$ between $\triangle_i$ and $\square_i$ where $\Phi_i(F_i)$. These mediators have the same underlying pure bigraph since $\triangle^{\mathrm{u}}$ is a pushout. By Definition 26, there is a unique mediator $F : I \to J$ between the bounds. $\square$

Our definition of paired sorting is based on link sorted and place sorted bigraphs. These have a pure bigraphical structure. The definition and proofs are general enough to transfer to pairs of sortings on local bigraphs. However, our definitions above do not apply to a combination of local bigraphs and kind sorted bigraphs as their structures are different – the former have no edges. However, the definitions and proofs should hopefully adapt to this situation quite easily. In general, place-sortings and local bigraphs should mix well as local bigraphs only change the link graph structure.

[7] They share the same set of controls and agree on arity, binding, *etc.*

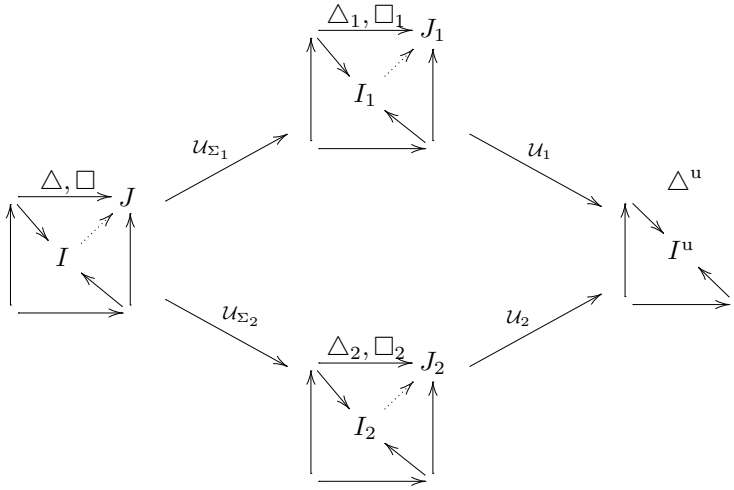


Figure 10: Pushouts are (weakly) reflected

Definition 27 (Rigid control-sorting [4])**.** *A place-sorting* $\Sigma_{\mathrm{K}} = (\mathcal{K}, \Theta, \Phi)$ *is a* rigid control-sorting *if it is equipped with a predicate* ϕ *and*

- $\Theta \subseteq \mathcal{P}(\mathcal{K})$.

Then Φ *requires for all bigraphs* G *that:*

P1 *if* $r = G^*(s)$ *then* $sort(s) = sort(r)$*;*

P2 *if* $r = G^*(v)$ *then* $\phi(ctrl_G(v), sort(r))$*;*

where r *is a root,* s *a site,* v *is a node, and* G^* *is the function which takes each site or node of* G *to the unique root above it.*

Definition 28 (kind rigid control-sorting [4])**.** *A place-sorting* $\Sigma_{\mathrm{K}} = (\mathcal{K}, \Theta, \Phi)$ *is a* kind rigid control-sorting *if it is defined over a kind signature and with a predicate* ϕ *and*

- $\Theta = \mathcal{P}(\mathcal{K}_{\mathsf{vis}}) \times \Theta_2$ *where* $\Theta_2 \subseteq \mathcal{P}(\mathcal{K})$.

Then Φ *requires for all bigraphs* G *that:*

K1 *if* $p = G(v)$ *then* $ctrl(v) \in sort_1(p)$*;*

K2 *if* $p = G(s)$ *then* $sort_1(s) \subseteq sort_1(p)$*;*

K3 *if* $sort_1(v) = \emptyset$*,* v *has no children;*

P1 *if* $r = G^*(s)$ *then* $sort_2(s) = sort_2(r)$*;*

P2 *if* $r = G^*(v)$ *then* $\phi(ctrl_G(v), sort_2(r))$*;*

where p *is a root or node,* $sort_1(v)$ *is the sort of a node,* r *is a root,* s *a site,* v *is a node, and* G^* *is the function which takes each site or node of* G *to the unique root above it.*

Kind rigid control-sorting is a paired place-sorting of kind sorting and rigid control-sorting. We have the following corollary of Proposition 18.

Corollary 20. *Kind rigid control-sorting creates RPOs.*

We can see that kind rigid control-sorting is a generalisation of both sortings. Kind sorting is the special case where $|\Theta_2| = 1$ and ϕ is always true (one particular example is $\Theta_2 = \mathcal{P}(\mathcal{K})$, $\phi(k, K) = k \in K$). Rigid control-sorting is the special case where all controls are visible, each non-atomic control receives sort $\mathcal{P}(\mathcal{K})$, each atomic control sort $\emptyset$[8].

It is important to consider which sortings to combine. The pointwise nature of our combination allows us to combine sortings with *conflicting* conditions. This will reduce the allowable bigraphs which may or not be what is wanted.

We finally note that the combination of kind sorting and rigid control-sorting is not ideal due to this pointwise combination. It would make sense to add another rule to Φ

$$\textbf{KP1} \qquad sort_1(t) \subseteq sort_2(t)$$

where t is a site or root of the bigraph G. This would eliminate some arrows of the s-category whose kind interface sort has some useless elements. It forms a subcategory of our combination. We have not taken this approach here as we have seen that forming subcategories can break RPO creation. It is likely that this extra rule is well behaved but it needs to be proven.

B.3 Location-aware Printing System

Corollary 20 allows us to combine kind sorting with Plato-graphical sorting (a special case of rigid control-sorting) and maintain RPO creation. We believe that this may be useful to model some context-aware systems that the latter sorting has been designed for. For example, the sort of a place of a kind sorted interface specifies which types of controls the place can contain when it is a root of some bigraph. This allows extra expressivity in parametric reaction rules – as the sites are sorted, the omission of a control in a site's sort guarantees that no parameter which will be placed in the hole has an exposed node with that control *i.e.* we can specify the absence of certain controls.

Another useful side-effect of the combination is that we remove more "junk" bigraphs which do not match real-life models.

As a motivating example, we present a kind sorting of the signature of the Location-aware Printing System [4] in Figure 11[9]. As an example of a use of the expressivity of kind parametric reaction rules, consider the rule (5') below that Birkedal et al. present for the Location-aware Printing System

$$\begin{aligned}&\mathbf{jobs}(\mathbf{doc}_{z,x}|-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev_x}|\mathbf{prt}_y(\mathbf{pcl})) \longrightarrow \\ &\qquad \mathbf{jobs}(-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl}|\mathbf{dat}_z)).\end{aligned} \qquad (5')$$

This rule models a rule where a print job ($\mathbf{doc}_{z,x}$) in a pool of pending jobs ($\mathbf{jobs}$) is not sent to a printer ($\mathbf{prt}_y$) until the printer and the device which submitted the job ($\mathbf{dev}_x$) are colocated. In the reactum, the print job has been sent to the printer as $\mathbf{dat}_z$. The first change we will make is trivial, we will just add a site into the rule.

$$\begin{aligned}&\mathbf{jobs}(\mathbf{doc}_{z,x}|-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev_x}|\mathbf{prt}_y(\mathbf{pcl})|-_1) \longrightarrow \\ &\qquad \mathbf{jobs}(-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl}|\mathbf{dat}_z)|-_1).\end{aligned} \qquad (5'_2)$$

[8] This retrieval of rigid control-sorting is suboptimal as many bigraphs with different kind interface sorts correspond to one rigid control-sorted bigraph. If the full subcategory of a kind s-category with the one interface sort $\mathcal{P}(\mathcal{K})$ creates RPOs – and it probably does in this case – this would yield a better match.

[9] We use kind sorting with capacities here which we hope to prove has RPOs in the future.

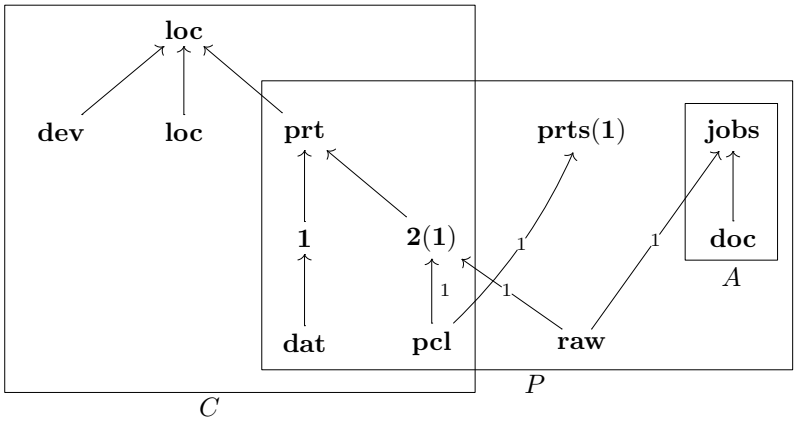


Figure 11: Kind sorting for the Location-aware Printing System

In Ireland, we suffer from an odd time dilation phenomenon – in our university, most print jobs are submitted to a printer hours before the guilty party arrives to extract them from the now treacherous mountain of paper. People get hurt. Rules $(5')$ and $(5'_2)$ above could be useful to model a system where the users, modelled with **dev** controls, are somehow linked to the documents they submit. The documents are then printed when the user enters a printer room (this could create a queue of people but maybe we could include a coffee machine and some biscuits in the model).

This brings up the question of what to do with printers when there are no users present. Perhaps we would like to switch them off when there are no users present and wake them up when one arrives (assume a hardware controller exists for this purpose). In the following, we write $\boxed{\neg K}$ for a site where the kind interface sort does not include the control K.

Assume a control $\mathbf{prt}^{\mathrm{off}}$ with the same kind sort as **prt** which models a printer which is switched off (but whose network card is still receiving power). The rule for putting printers to sleep (one by one) can be written as:

$$\mathbf{loc}(\mathbf{prt}_y(-_0)|\boxed{\neg\mathbf{dev}}\,_1) \longrightarrow \mathbf{loc}(\mathbf{prt}^{\mathrm{off}}_y(-_0)|\boxed{\neg\mathbf{dev}}\,_1).$$

The rule for waking the printers up (again, one by one) is:

$$\mathbf{loc}(\mathbf{prt}^{\mathrm{off}}_y(-_0)|\mathbf{dev}_x|-_1) \longrightarrow \mathbf{loc}(\mathbf{prt}_y(-_0)|\mathbf{dev}_x|-_1).$$

These two rules demonstrate how kind sorting partially answers the problem mentioned by Birkedal et al., that "it is generally very difficuly... to observe the *absence* of something in the context directly." The kind sorts of sites can express which controls may *not* be placed directly under the site in a composition.

B.4 "Find them and kill them"

Imagine a bigraphical system consisting of a nesting of locations and devices where we want to ask "what devices are present within location l." There is no simple, internal way to answer this question using the theory. However, Birkedal et al. have presented a set of bigraphical rules which implement the query [3]. It is a very nice example of a bigraphical tree-traversal 'algorithm.' It works by using a pair of controls **f** and **s** to traverse a tree of locations and one-by-one find a device and add it to the output of the query.

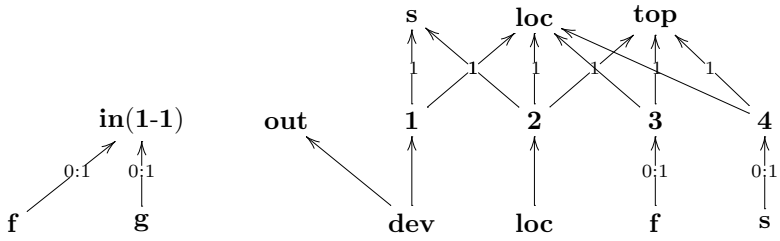


Figure 12: Kind sorting for the "copy all devices" query

We can use their technique to define a similar algorithm to implement an operation which finds and moves (or copies or deletes) all devices in a building to an output location. In the context of their model, this set of rules does not make much sense (unless we consider it to represent "throw all PDAs and PCs out the window" which is certainly tempting) but we introduce it to demonstrate how certain operations can be done in one step using a kind sorting and Milner's idea of multi-nodes. Further, their example was based on the assumption that locations can only contain either devices or other locations – this assumption can be enforced using a kind sorting[10].

The kind sorting (with capacities) in described in Figure 12 with locations **loc** and devices **dev**. The multi-node **top** (defined like the **loc** except that it is missing the **dev** section) represents a top location. The two controls **in** and **out** represent input and output nodes. The control **g** is a dummy control used to represent that a query operation is in progress. The control **f** is used to mark the location where the algorithm is currently operating. The control **s** is used to keep track of the locations in the tree which have been visited by collecting them.

Combining multi-nodes with kind sorting allows us to separate **loc** and **s** nodes into areas which can each contain one type of control. If a model required locations to contain other types of controls, more sections could be added to the multi-nodes. We consider multi-node ions as having multiple sites, ordered in the obvious way. This example requires kind sorting with capacities the theory of which we still need to investigate.

Multi-nodes can be written as *e.g.* $\mathbf{loc}_x(-_0 \parallel -_1 \parallel -_2 \parallel -_3)$. When a section of a multi-node is empty, we denote this with blank space as usual so $\mathbf{loc}_x(\ \parallel\ \parallel\ \parallel\ )$ denotes a multi-node **loc** with no contents.

[10] In fact, 'real-life' models like this were our motivation to explore kind sortings.

The set of rules for the copy algorithm is as follows:

$$\begin{array}{c}\text{initialise}\\ \mathbf{in}(\mathbf{f}) \parallel \mathbf{top}(-_0 \parallel \ \parallel \ ) \parallel \mathbf{out}() \\ \longrightarrow \mathbf{in}(\mathbf{g}) \parallel \mathbf{top}(-_0 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel \ )) \parallel \mathbf{out}() \\ \text{dig from top} \\ \mathbf{top}(\mathbf{loc}_x(-_1 \parallel -_2 \parallel \ \parallel \ ) \mid -_0 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel -_3)) \\ \longrightarrow \mathbf{top}(\mathbf{loc}_x(-_1 \parallel -_2 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel \ )) \mid -_0 \parallel \ \parallel \mathbf{s}(\ \parallel -_3)) \\ \text{copy colocated devices} \\ \mathbf{loc}_x(-_0 \parallel -_1 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel -_2)) \parallel \mathbf{out}(-_3) \\ \longrightarrow \mathbf{loc}_x(\ \parallel -_1 \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_2)) \parallel \mathbf{out}(-_3 \mid -_0) \\ \text{dig} \\ \mathbf{loc}_x(\ \parallel \mathbf{loc}_y(-_3 \parallel -_4 \parallel \ \parallel \ ) \mid -_0 \parallel \mathbf{f} \parallel \mathbf{s}(-_1 \parallel -_2)) \\ \longrightarrow \mathbf{loc}_x(\ \parallel \mathbf{loc}_y(-_3 \parallel -_4 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel \ )) \mid -_0 \parallel \ \parallel \mathbf{s}(-_1 \parallel -_2)) \\ \text{climb} \\ \mathbf{loc}_x(\ \parallel \mathbf{loc}_y(\ \parallel \ \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_1)) \mid -_2 \parallel \ \parallel \mathbf{s}(-_3 \parallel -_4)) \\ \longrightarrow \mathbf{loc}_x(\ \parallel -_2 \parallel \mathbf{f} \parallel \mathbf{s}(-_3 \parallel \mathbf{loc}_y(-_0 \parallel -_1 \parallel \ \parallel \ ) \mid -_4)) \\ \text{climb to top} \\ \mathbf{top}(\mathbf{loc}_x(\ \parallel \ \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_1)) \mid -_2 \parallel \ \parallel \mathbf{s}(\ \parallel -_3)) \\ \longrightarrow \mathbf{top}(-_2 \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel \mathbf{loc}_x(-_0 \parallel -_1 \parallel \ \parallel \ ) \mid -_3)) \\ \text{clean up} \\ \mathbf{in}(\mathbf{g}) \parallel \mathbf{top}(\ \parallel \mathbf{f} \parallel \mathbf{s}(\ \parallel -_0)) \\ \longrightarrow \mathbf{in}() \parallel \mathbf{top}(-_0 \parallel \ \parallel \ )\end{array}$$

The algorithm implements a copy operation where the hierarchy is not copied (a 'flattened copy'). Instead of taking copying one device at a time as would be done in pure bigraphs, the fact that a site only contains **dev** controls means that we can copy the entire site to the output node. Using kind sorting with capacaties, we can avoid copying empty sites in the copy rule (which would otherwise create infinite loops) by requiring that the site contains a minimum of one control.

We can describe when to apply the rules from the point of view of **f** as below.

Rule	**Do when**
initialise	query is pending
dig from top	at top, locations here
copy colocated devices	not at top, devices here
dig	not at top, no devices here, locations here
climb	not at top, no devices here, no locations here, location above
climb to top	below top, no devices here, no locations here
clean up	at top, no locations here

This table displays how the rules fire under separate criteria and is why we call the set of rules an algorithm. The algorithm is non-deterministic as the digging rules allow any colocated location to be explored.

The sequence in Figure 13 shows this algorithm at work on the tree

$$\mathit{InitialTree} = \mathbf{top}(\mathbf{loc}(\mathbf{dev}_1 \parallel \mathbf{loc}(\mathbf{dev}_2) \mid \mathbf{loc}(\mathbf{dev}_3 \mid \mathbf{dev}_4)) \mid \mathbf{loc}()).$$

$$
\begin{aligned}
&\mathbf{in}(\mathbf{f}) \parallel \mathit{InitialTree} \parallel \mathbf{out}() \\
&= \mathbf{in}(\mathbf{f}) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \mid \mathbf{loc}()) \parallel \mathbf{out}() \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \mid \mathbf{loc}() \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{out}() \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \mid \mathbf{loc}(\mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}()) \parallel \mathbf{out}() \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}() \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}) \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}() \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{d_1})) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d_2} \parallel \mathbf{f} \parallel \mathbf{s}()) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}) \parallel \mathbf{s}(\mathbf{d_1})) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d_2})) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}) \parallel \mathbf{s}(\mathbf{d_1})) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4} \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d_3} \mid \mathbf{d_4})) \parallel \mathbf{s}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}|\mathbf{d_3}|\mathbf{d_4}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4}))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}|\mathbf{d_3}|\mathbf{d_4}) \\
&\to \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{f} \parallel \mathbf{s}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \mid \mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}|\mathbf{d_3}|\mathbf{d_4}) \\
&\to \mathbf{in}() \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d_1} \parallel \mathbf{loc}(\mathbf{d_2}) \mid \mathbf{loc}(\mathbf{d_3} \mid \mathbf{d_4})) \mid \mathbf{loc}()) \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}|\mathbf{d_3}|\mathbf{d_4}) \\
&= \mathbf{in}() \parallel \mathit{InitialTree} \parallel \mathbf{out}(\mathbf{d_1}|\mathbf{d_2}|\mathbf{d_3}|\mathbf{d_4})
\end{aligned}
$$

Figure 13: Example run of the flattened copy operation

We unambiguously omit the names and empty sections of multi-nodes to make the example easier to read and colour the node currently holding **f** in blue. We write **dev** as **d** to save space. When following the sequence, it may help to recognise that according to the rules, an **f** node can only dig or climb if there are no colocated devices and can only climb to the top if there are no colocated devices or locations. This expressivity in the rules comes from the use of the specific kind sorted multi-nodes **loc**, **top**, and **s** where each section contains a unique type of control.

One assumption we have made in the rules of the algorithm is that there is one f query propagating through the hierarchy. The rules could be modified to allow simultaneous queries but these may interfere with each other. A possible solution is to add capacities to rigid control-sorting which would allow us to specify that only one f query exists below a given root and also use a link sorting where inputs and outputs are linked to at most one query. Because the clean up rule destroys an f agent, we would need to be able to specify 'at most one f agent' rather than 'exactly one f agent.' This relaxed sorting would probably not reflect pushouts (similarly to kind sortings, minimal interfaces are not enforced) but may still weakly reflect them.

C Typed links

In this section we will present a link sorting for local bigraphs which creates RPOs. The elements of the set of sorts are called *types* as they are used to model simple types. The sorting has the simple condition that points are connected to links with the same type.

We start by repeating Leifer and Milner's definition of link sorting, adding some notation for convenience.

Definition 29 (sorted link graphs [21])**.** *A signature $\mathcal{K}$ is* Θ-sorted *if it is enriched by a sorting $sort_K : arity(K) \to \Theta$ for each control K. We abbreviate $sort_K(p)$ to $sort(p)$ when K is understood or irrelevant. We write $\mathcal{K}^\tau$ for a signature $\mathcal{K}$ equipped with a sorting for each control. An interface X is Θ-sorted if it is enriched by ascribing a sort $sort(x)$ to each name $x \in X$.*

A link graph is Θ-sorted over $\mathcal{K}$ *if its interfaces are Θ-sorted, and for each K, i the sort $sort_K(i)$ is ascribed to the i^{th} port of every K-node.*

$\textsc{Lig}(\Theta, \mathcal{K})$ denotes the monoidal precategory of sorted link graphs whose identities, composition and tensor product are defined in the obvious way in terms of the underlying (unsorted) link graphs.

Definition 30 (link sorting [21])**.** *A* (link)-sorting (discipline) *is a triple*

$$\Sigma = (\mathcal{K}, \Theta, \Phi)$$

where $\mathcal{K}$ is Θ-sorted, and Φ is a condition on Θ-sorted link graphs over $\mathcal{K}$. The condition Φ must be satisfied by the identities and preserved by both composition and tensor product.

A link graph in $\textsc{Lig}(\Theta, \mathcal{K})$ is said to be Σ-sorted *if it satisfies Φ. The Σ-sorted link graphs form a monoidal sub-precategory of $\textsc{Lig}(\Theta, \mathcal{K})$ denoted by $\textsc{Lig}(\Sigma)$. Further, if $\mathcal{R}$ is a set of Σ-sorted reaction rules then $\textsc{Lig}(\Sigma, \mathcal{R})$ is a* Σ-sorted *LRS.*

Definition 31 (simply typed link sorting)**.** *In a* simply typed link sorting *$\Sigma = (\mathcal{K}, \Theta, \Phi)$, Θ is some arbitrary set whose elements are called* types *and $\mathcal{K}$ is a Θ-sorted binding signature. The condition Φ is:*

- *if $link(p) = l$ then $sort(p) = sort(l)$*

for points p and links l.

This sorting extends to local bigraphs [27] in the obvious manner and is satisfied by identities and preserved by composition and tensor product. We denote the s-category of simply typed link sorted local bigraphs over a signature $\mathcal{K}$ as $\textsc{SBG}_{\text{loc}}(\mathcal{K}^\tau)$. The forgetful functors

$$\textsc{SBG}_{\text{loc}}(\mathcal{K}^\tau) \xrightarrow{\mathcal{U}_{\text{type}}} \textsc{BG}_{\text{loc}}(\mathcal{K}) \xrightarrow{\mathcal{U}_{\text{loc}}} \textsc{BG}_{\text{pure}}(\mathcal{K}^{\text{u}})$$

respectively forget sorting and locality of interfaces. See Milner's work [27] for a definition of $\mathcal{U}_{\text{loc}}$.

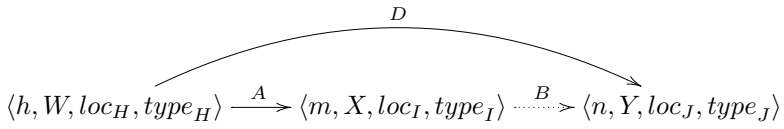
Notation. *From now on any mention of sorting implies a simply typed link sorting and sorted bigraphs refer to local bigraphs sorted with a simply typed link sorting in an arbitrary s-category of sorted bigraphs. When discussing sorted bigraphs, we write $type(q) = sort(q)$ where q is a port or a name. A local interface sorted over $\Sigma = (\mathcal{K}, \Theta, \Phi)$ is written as $I = \langle m, X, loc, type_I \rangle$ where $type : X \to \Theta$ assigns types to names.*

The underlying local bigraph *of a sorted local bigraph G is defined as $\mathcal{U}_{\text{type}}(G)$. The* underlying pure bigraph *of a sorted local bigraph G is defined as $(\mathcal{U}_{\text{loc}} \circ \mathcal{U}_{\text{type}})(G)$ and denoted G^{u}.*

The parallel product of two sorted local interfaces is defined when the union of their type maps is single-valued. The parallel product of two sorted local bigraphs G and G' is defined when the parallel product of their interfaces and $\mathcal{U}_{\text{type}}(G) \,\|\, \mathcal{U}_{\text{type}}(G')$ are. Essentially, parallel product respects typing.

Lemma 21. *Given two sorted bigraphs $D : H \to J$ and $A : H \to I$ where A has no idle links, let $B' : \mathcal{U}_{\text{type}}(I) \to \mathcal{U}_{\text{type}}(J)$ be a local bigraph such that $\mathcal{U}_{\text{type}}(D) = B' \circ \mathcal{U}_{\text{type}}(A)$. Then there exists a unique sorted bigraph $B : I \to J$ such that $\mathcal{U}_{\text{type}}(B) = B'$ and $D = B \circ A$.*

Proof. The situation is depicted below.



Define B as B' with inner interface I and outer interface J. We will prove that B is well-sorted.

Let $x \in X$ and $link_B(x) = l$ for some link l of B. x is not idle in A so there exists a point p in A such that $link_A(p) = x$ and so $link_D(p) = l$. As A and D are sorted, we conclude that $type(p) = type(x) = type(l)$. Therefore, B is well-sorted on inner names.

Let p be a port of B and $link_B(p) = l$ for some link l of B. Then $link_D(p) = l$ and $type(p) = type(l)$. Therefore, B is also well-sorted on ports and hence well-sorted.

We have a bigraph defined $B : I \to J$ such that $\mathcal{U}_{\text{type}}(B) = B'$. As $\mathcal{U}_{\text{type}}$ is faithful, B is unique. □

In the following, i ranges over $\{0, 1\}$ and $\bar{\imath} = 1 - i$.

Construction 2.

Let $\vec{A} : H \to \vec{I}$ have a bound $\vec{D} : \vec{I} \to J$ in a s-category of sorted bigraphs with $H = \langle h, W, loc_H, type_H \rangle$, $I_i = \langle m_i, X_i, loc_i, type_i \rangle$, and $J = \langle n, Y, loc_J, type_J \rangle$. We construct an RPO $(\vec{B}, B)$ for $\vec{A}$ to $\vec{D}$ as follows.

First build a local RPO $(\vec{B}_{\text{loc}}, B_{\text{loc}})$ for $\mathcal{U}_{\text{type}}(\vec{A})$ to $\mathcal{U}_{\text{type}}(\vec{D})$ where the inner face of B is $\langle m, X, loc_I \rangle$.

The set X is defined as in the pure construction [17] for link graph RPOs as follows. Let V_i be the nodes of A_i, $V_2 = V_0 \cap V_1$, and $V_{\bar{\imath}} - V_2 \uplus V_3$ be the nodes of D_i. Edges E_i (recovered from $\mathcal{U}_{\text{loc}} \circ \mathcal{U}_{\text{type}}$) are treated similarly and ports P_i are treated like their nodes. First, they define:

$$X_i' \stackrel{\text{def}}{=} \{x \in X_i \mid D_i(x) \in E_3 \uplus Y\}.$$

Next, they define $\cong$ to be the smallest equivalence on the disjoint sum $X_0' + X_1'$ for which

$$(0, x_0) \cong (1, x_1) \text{ whenever } A_0(p) = x_0 \text{ and } A_1(p) = x_1 \text{ for some } p \in W \uplus P_2$$

and define $X \stackrel{\text{def}}{=} (X_0' + X_1')/\cong$. For each $x \in X_i'$ the $\cong$-equivalence class of (i, x) is denoted by $\widehat{i, x}$.

We define the function $type_I$ as $type(\widehat{0, x_0}) = type(x_0)$. $type_I$ is well-defined as follows. If $(0, x_0) \cong (1, x_1)$ then $A_0(p) = x_0$ and $A_1(p) = x_1$. Since A_0 and A_1 are sorted, $type(p) = type(x_0) = type(x_1)$. This implies that all $\cong$-equivalent names x_i have the same type. The sorted RPO is then defined as $(\vec{B}_{\text{loc}}, B_{\text{loc}})$ lifted to the sorted setting and with the inner interface of B defined as $\langle m, X, loc_I, type_I \rangle$.

□

Proposition 22 (valid RPO construction)**.** *Construction 2 builds RPOs in ´*SBG_{loc}.

Proof. We first prove that the construction yields a sorted triple $(\vec{B}, B)$. We do this by referring to the pure construction of link graph RPOs [17].

We first prove that B_0 is well-sorted. The proof for B_1 is similar. We break the proof over the cases in the construction.

- Let $x \in X_0$.
  - If $x \in X_0'$ then $B_0(x) = \widehat{0, x}$ and $type(x) = type(\widehat{0, x})$ by construction.
  - If $x \notin X_0'$ then $B_0(x) = p_1$ is a binding port where $p_1 \in P_1 - P_2$. Then $D_0(x) = p_1$ and since D_0 is sorted, $type(x) = type(p_1)$.
- Let $p \in P_1 - P - 2$.
  - If $B_0(p) = \widehat{1, x}$ then $A_1(p) = x$. As A_1 is sorted, we have $type(p) = type(x) = type(\widehat{1, x})$ from the construction.
  - If $B_0(p) = p'$ then $D_0(p) = p'$ and as D_0 is sorted, $type(p) = type(p')$.

Neither B_0 nor B_1 have idle names. Therefore, by Lemma 21, B is well-sorted. Therefore $(\vec{B}, B)$ is a relative bound.

Let $(\vec{C}, C)$ be a candidate RPO with mediating interface K. There is a unique mediator $\mathcal{U}_{\text{type}}(K) : \mathcal{U}_{\text{type}}(I) \to \mathcal{U}_{\text{type}}(C)$. As neither B_0 nor B_1 have idle names, there is then a unique sorted mediator $K : I \to C$ by Lemma 21. Hence, Construction 2 builds an RPO. □

D Homomorphic sortings

A homomorphic sorting was successfully used to model finite CCS [26]. In this appendix, we describe why a homomorphic sorting cannot be used to properly encode $\Lambda^{\tau}_{\mathrm{sub}}$ using our signature (Definition 11). We must emphasise that this is not surprising – homomorphic sorting was introduced and used to model finite CCS which has a simple sorting and introducing a stronger sorting would have been unnecessary – but we believe that the next section may serve to better explain kind sorting.

D.1 Homomorphic sortings as kind sortings

A homomorphic sorting is a type of place-sorting [26]. It has the property that the children of a root or node all have the same sort and, further, a root has the same sort as all of its children. We present the definition slightly differently – and worse – here to fit our previous definitions.

Definition 32 (homomorphic signature)**.** *A* [homomorphic signature $\{\mathcal{K}, \Theta, \mathit{arity}, \mathit{actv}, \mathit{kind}, \mathit{sort}, \mathit{prnt}\}$ *is composed of a set* $\mathcal{K}$ *of* controls*, a set* Θ *of sorts, and five maps:*

$$\begin{array}{rcl@{\qquad}rcl} \mathit{arity} & : & \mathcal{K} \to \mathbb{N} & \mathit{sort} & : & \mathcal{K} \to \Theta \\ \mathit{actv} & : & \mathcal{K} \to \{\mathsf{passive}, \mathsf{active}\} & \mathit{prnt} & : & \Theta \to \Theta \\ \mathit{kind} & : & \mathcal{K} \to \{\mathsf{atomic}, \mathsf{non\text{-}atomic.}\} & & & \end{array}$$

where atomic controls must be passive. Without loss of generality, we will assume sort is surjective (a partition). We denote an arbitrary homomorphic sorting with $(\mathcal{K}, \Theta)$ *with assumed functions* $\mathit{arity}, \mathit{actv}, \mathit{kind}, \mathit{sort}$*, and* prnt*.*

Definition 33 (homomorphic grouping)**.** *Let* $(\mathcal{K}, \Theta)$ *be a homomorphic signature. For each control* $K \in \mathcal{K}$*, the set* $(\mathit{sort}^{-1} \circ \mathit{sort})(K)$*, the* homomorphic grouping *of* K*, is denoted by* Θ_K*.*

If K_1 and K_2 have the same sort then $\Theta_{K_1} = \Theta_{K_2}$.

Definition 34 (homomorphic sorting [26])**.** *A place-sorting* $\Sigma_{\mathrm{K}} = (\mathcal{K}, \Theta, \Phi)$ *over a homomorphic signature* $(\mathcal{K}, \Theta)$ *is a* homomorphic sorting *if for each site or node* w *in a bigraph* G*:*

- *if* $G(w) = v$ *then* $\mathit{sort}(v) = \mathit{prnt}(\mathit{sort}(w))$*;*
- *if* $G(w) = r$ *then* $\mathit{sort}(r) = \mathit{sort}(w)$*.*

where v *is a node and* r *is a root.*

$$\mathit{srt}(\mathit{parent}(v)) = \mathit{prnt}(\mathit{sort}(v)) \qquad \{v' \mid \mathit{parent}(v') = v\} \subseteq \mathit{knd}(v)$$

Figure 14: Containment conditions of homomorphic sortings and kind sortings

Homomorphic sorting can be used to encode finite CCS in bigraphs [26]. It may be described as a kind sorting. Figure 14 may help to illustrate this by summarising the main conditions on each sorting (*parent* denotes the parent map of the place graph). Note that for each non-atomic $K \in \mathcal{K}$, the set $(sort^{-1} \circ prnt^{-1} \circ sort)(K)$ is the set of controls which nodes of K may be a parent of according to the homomorphic sorting.

The following definition fixes a mistake in previous work [9] where our interpretation of homomorphic sortings as kind sortings only worked for homomorphic sortings where $prnt : \Theta \to \Theta$ is injective.

Definition 35 (homomorphic kind signature)**.** *Let* $\{\mathcal{K}, \Theta, arity, actv, kind, sort, prnt\}$ *be a homomorphic signature. The corresponding* homomorphic kind signature $\{\mathcal{K}, arity_{\mathsf{h}}, actv_{\mathsf{h}}, vsbl_{\mathsf{h}}, kind_{\mathsf{h}}\}$ *is the kind signature where:*

- $arity_{\mathsf{h}} = arity$, $actv_{\mathsf{h}} = actv$;
- $vsbl_{\mathsf{h}}$ *is the constant function assigning* vis *to each control;*
- $kind_{\mathsf{h}}$ *is defined by:*

$$\begin{array}{lcll} kind_{\mathsf{h}}(K) & = & (sort^{-1} \circ prnt^{-1} \circ sort)(K) & \text{if } kind(K) = \mathsf{non\text{-}atomic}, \\ kind_{\mathsf{h}}(K) & = & \emptyset & \text{if } kind(K) = \mathsf{atomic}. \end{array}$$

The associated kind sorting satisfies most of the conditions of a homomorphic sorting. The final condition is that the roots are sorted. For this, we use the homomorphic s-category associated with the sorting (where the interface sorts are the homomorphic groupings of the signature).

D.2 Homomorphic sorting and $\Lambda^{\tau}_{\text{sub}}$

One property we want of our model is that typing is somehow preserved (Proposition **??**). We cannot achieve this with homomorphic sorting.

Example 5 ($\mathcal{K}^{\tau}$ cannot be sorted homomorphically for our purposes)**.** We will try to homomorphically sort $\mathcal{K}^{\tau}$. Give var^G the sort θ_1 for some A. In our model, var^G should be able to be a child of both 2^G and def^G. According to the definition of homomorphic sorting, both 2^G and def^G must have the same sort θ_2. Let $\mathsf{app}^{G \to G, G}$ have the sort θ_3. In our model, $\mathsf{app}^{G \to G, G}$ should be able to contain 2^G. However, as 2^G and def^G have the same sort, the homomorphic sorting allows def^G to be a child of $\mathsf{lam}^{G \to G}$. This breaks the sorting of the controls (Definition 12) leading to bigraphs which do not represent $\Lambda^{\tau}_{\text{sub}}$ terms (this will happen anyway *e.g.* with non-prime bigraphs, but we wish to restrict it as much as possible).

The problem gets worse. Let $G' \neq G$. In our model, var^G should be able to be a child of $\mathsf{lam}^{G' \to G}$ as well. Therefore, the sort of $\mathsf{lam}^{G' \to G}$ is also θ_2. This means that using the homomorphic sorting allows $\mathsf{lam}^{G' \to G}$ to be a child of $\mathsf{app}^{G \to G, G}$ which does not respect the typing of $\Lambda^{\tau}_{\text{sub}}$.

The problem with using homomorphic sorting for our model is that the sorts are merged – degeneracy creeps in. In the example above, the function $prnt : \Theta \to \Theta$ is (partially) defined as

$$prnt(\theta_1) \mapsto \theta_2, \quad prnt(\theta_2) \mapsto \theta_3, \quad prnt(\theta_3) \mapsto \theta_2.$$

This looping structure ($\theta_2 \mapsto \theta_3 \mapsto \theta_2$) comes from *prnt* being an endomap (see Lawvere and Schanuel's book [20] for some graphical intuitions). Endomaps can be seen as a special sort of restricted directed graph. Kind sortings are essentially relations and relations may be described by directed graphs. This intuition fits the fact that homomorphic sortings are special cases of kind sortings [9]. Put another way, homomorphic sortings are based on functions whereas kind sortings are based on relations.

E Typing $'\Lambda_{\text{BIG}}$

Our approach in modelling simply typed Λ_{sub} was to use sortings of bigraphs. Here, we present another idea. We extend interfaces with types $A \in \tau$ and use the rules of Figure 15 – based on those of Figure 1 – to identify prime bigraphs which are related to simply typed Λ_{sub} terms.

Using the rules, we have something approaching a static correspondence. If $X \supseteq \mathsf{FV}(t)$ and $\Gamma \vdash t : A$ then we can build a corresponding $'\Lambda_{\text{BIG}}$ term of type A and, repeatedly applying the first rule, with a set of idle names $X \setminus \mathsf{FV}(t)$.

For dynamic correspondence we require that if $\Gamma \vdash t : A$ and $t \longrightarrow_{\text{ACD}} t'$ then if G is an encoding of t with type A, $G \twoheadrightarrow G'$ with G' an encoding of t' with type A. As reduction can lose variables, the first rule in the table allows these to be added as idle names.

The rules below cannot derive parametric redexes or reactums, only prime bigraphs. However, the reaction relation is defined on ground rules so the rules seem a good match. Rules to allow λ-contexts could be similarly defined.

$$\frac{G : I \to \langle 1, X, A\rangle}{(\mathrm{id} \oplus \{x : A\}) \circ G : I \to \langle 1, X \uplus \{x : A\}, A\rangle} \qquad \frac{}{\mathsf{var}_{x:A} : \epsilon \to \langle 1, \{x : A\}, A\rangle}$$

$$\frac{G : I \to \langle 1, X \uplus \{x : A\}, B\rangle}{(\mathsf{lam}_{(x:A)} \oplus \mathrm{id}_X) \circ G : I \to \langle 1, X, A \to B\rangle}$$

$$\frac{F : I \to \langle 1, X, A \to B\rangle \qquad G : J \to \langle 1, Y, A\rangle}{(\mathsf{app} \oplus (X|Y))(F \parallel G) : I \parallel J \to \langle 1, X \cup Y, B\rangle}$$

$$\frac{F : I \to \langle 1, X \uplus \{x : A\}, B\rangle \qquad G : J \to \langle 1, Y, A\rangle}{(\mathsf{sub}_{(x:A)} \oplus \mathrm{id}_X)(F \mid (\mathsf{def}_{x:A} \oplus \mathrm{id}_X)G) : I \mid J \to \langle 1, X \cup Y, B\rangle}$$

Figure 15: Typing rules for $'\Lambda_{\text{BIG}}$