A Bigraphs Paper of Sorts*

Blair Archibald[1 [0000−0003−3699−6658]] and Michele Sevegnani[1 [0000−0001−6773−9481]]

School of Computing Science, University of Glasgow, UK
{blair.archibald,michele.sevegnani}@glasgow.ac.uk

Abstract. Bigraphs are an expressive graphical modelling formalism to represent systems with a mix of both spatial and non-local connectivity. Currently it is possible to write nonsensical models, *e.g.* with a Room nested inside a Person rather than Person nested inside a Room, or to create a hyperedge from what should be a binary link. A sorting scheme can be used to filter badly-formed bigraphs from those that are well formed. While the theory of bigraph sorts is well developed, none of the existing methods leads to a practical implementation. Instead they are based on tables of descriptions or semi-mathematical notations. We look at sorting bigraphs through a practical lens: developing a new sorting language, and show how an extension to the existing theory of bigraphs, in the form of well-sorted interfaces, paves the way for an implementation of well-sorted bigraphs. We discuss the trade-offs of this approach, and show how it allows sorts to be specified for existing bigraph models found in the literature.

Keywords: Bigraphs · Sortings · Type systems.

1 Introduction

Bigraphs [19] are an expressive graphical modelling formalism designed to represent systems that have both spatial aspects, *e.g.* a Person in a Room, and non-local aspects, *e.g.* communication between Person entities in (possibly) different Rooms. In fact, they are sometimes too expressive: allowing nonsense models to be created, *e.g.* where a Room is nested *inside* a Person! Sorting schemes [19] have been proposed as a way to filter badly formed models—in a similar way to how a type system excludes badly formed programs. In a sorting scheme, entities are assigned both a *type* (called a control), *e.g.* Person, and a *sort*, *e.g.* `moveable`, and a set of constraints determines how entities of different sorts may interact. For example we might constrain that `stationary` sorts are never nested below `moveable` to exclude our Room within a Person issue.

While there have been many theoretical discussions of sorting schemes [19, 18, 15, 6, 3, 9, 5], and they have been used to describe well-formed models of systems such as CCS, Petri-nets, and π-calculus, there is currently no practical

* Supported by an Amazon Research Award on Automated Reasoning.

Table 1: Example of sorting conditions for a bigraph encoding of the Actor Model recreated (partially) from [23]. Notation $\widehat{\texttt{be}}$ indicates sum sorts, *i.e.* `b` or `e`, and θ_{Act} is a set of sorts.

Constraint	Description
ϕ_1	all children of a θ-regions have sort θ, where $\theta \in \theta_{\text{Act}}$
ϕ_2	all children of an `a`-node have sort $\widehat{\texttt{be}}$

tooling available for computationally specifying or working with sorts. The lack of tooling is obvious from the current descriptions of sorting schemes which are often simply plain text descriptions of their expected operation. For example Table 1, partially recreated from [23], shows some of the types of constraints we would like to specify: child relationships (*e.g.* children of `x` has sort `y`), and cardinality constraints (*e.g.* `x` has one `z` child). While these textual descriptions are useful, they are not immediately amenable to automatic analysis, and it is up to the modeller to ensure these constraints are met.

In this paper, we unlock the potential for automated implementation of sort checking/inference by:

- Defining a simple, yet powerful, language for describing entities and their sorts.
- Showing how the standard operators for building bigraphs, *e.g.* written in terms of tensor products, compositions, and substitutions, can be extended to only allow building well-sorted bigraphs.
- Using models found in the literature, show how our language captures existing modelling domains.

This approach paves the way for an implementation of bigraph sorts, *e.g.* in BigraphER [22], but at present no implementation exists. Throughout we utilise a running example of Petri-nets [20] modelled in bigraphs.

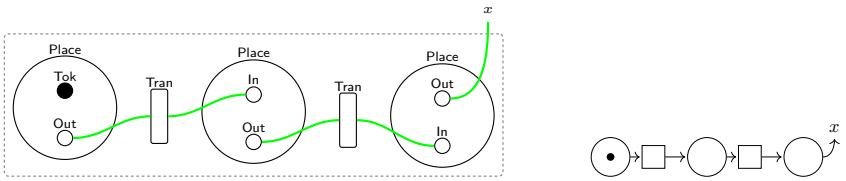
The paper is organised as follows: Section 2 gives necessarily background on bigraphs and sorting schemes. In Section 3 we show a new language for expressing sorts, and in Section 4 we show how this language enables sortings for elementary bigraphs (and therefore any bigraph). Section 5 applies the language to a set of applications. We discuss the limitations, future work, and conclusion in Section 6.

2 Background

2.1 Bigraphs

A bigraph consists of two orthogonal structures defined on the same set of entities: a *place graph* (a forest) that describes the *nesting* of entities, and a *link graph* that provides non-local hyperlinks between entities.

[1] Open meaning the output of this system can connect, via x, to another Petri net [4].



(a) (b)

Fig. 1: (a) Bigraph modelling an (open[1]) Petri net; (b) Petri net representation.

An example bigraph that models a Petri net is in Fig. 1. This differs from existing Petri net models [16], by allowing an unbounded number of tokens in each place and using extra entities for linking rather than defining a family of places/transitions (one for each linking structure, *e.g.* 1 link in, 2 links out). We draw entities as different (coloured) shapes. Containment illustrates the spatial nesting relationship, *e.g.* Tok is contained by Place, while green hyperedges (1-to-n links) represent non-spatial connections, in this case giving the wiring of places and transitions. Entities have a fixed *arity* (number of links/ports), *e.g.* Out has arity 1, but links may be disconnected/closed, *i.e.* a 1-to-0 link. We use *port* to refer to the link point of an entity, and link to mean a collection of ports (and names).

Each place graph has m *regions*, shown as the dashed rectangles, and n *sites*, usually shown as filled dashed rectangles. Regions represent parallel parts of the system, and sites represent abstraction, *i.e.* an unspecified bigraph (including the empty bigraph) exists there. Similarly, link graphs have a (finite) set of inner names and outer names, *e.g.* $\{x\}$.

Bigraphs are compositional structures, and we build larger bigraphs from smaller bigraphs. A bigraph is described by an interface $B : \langle n, X\rangle \to \langle m, Y\rangle$ specifying bigraph B maps n sites, to m regions, and inner names X to outer names Y. Composition of bigraphs, denoted $\circ$, consists of placing regions in sites (when $n = m$), and connecting inner and outer-faces on like-names. Composition combines bigraphs *vertically*, but we can also combine bigraphs *horizontally* through the tensor product $\otimes$. This tensor product extends both the sites/regions and name sets for the interfaces. $\otimes$ is only defined when the sets of interface names are disjoint. We introduce additional algebraic representations, *e.g.* symmetries, for bigraphs in the following sections.

2.2 Sorting

Given the term-rewriting/algebraic nature of bigraphs, initial sorting schemes were designed in a similar manner to many-sorted algebras but with more freedom due to the need to classify both places and links, and the flexibility of changing the argument order for constructors (to reflect the graph-like nature).

Milner describes a sorting scheme [19] based on an assignment of sorts to entities (controls) and a *formation rule* as in Table 1 that defines constraints on

$$
\begin{aligned}
\langle \text{sort} \rangle &::= \texttt{sort}\ \langle \text{sname} \rangle \ \mid \\
&\quad\ \texttt{sort}\ \langle \text{sname} \rangle\ \texttt{=}\ \langle \text{constructors} \rangle^{*}_{\mid} \\
\langle \text{constructors} \rangle &::= \langle \text{cname} \rangle [\{\langle \text{lpat} \rangle^{*}_{,}\}]\ [\langle \text{pat} \rangle] \\
\langle \text{pat} \rangle &::= \langle \text{pat} \rangle \times \langle \text{pat} \rangle \ \mid\ \langle \text{pat} \rangle\ \texttt{+}\ \langle \text{pat} \rangle \ \mid \\
&\quad\ \langle \text{pat} \rangle^{*} \ \mid\ \texttt{(}\langle \text{pat} \rangle\texttt{)} \ \mid\ \langle \text{baseS} \rangle \\
\langle \text{baseS} \rangle &::= \langle \text{sname} \rangle \ \mid\ \mathbb{1} \\
\langle \text{lpat} \rangle &::= \langle \text{sname} \rangle \to \langle \text{pat} \rangle \\
\langle \text{sname} \rangle &::= \texttt{[a-z][A-z0-9]+} \\
\langle \text{cname} \rangle &::= \texttt{[A-Z][A-z0-9]+}
\end{aligned}
$$

Fig. 2: Grammar for sort and entity specification. Expressions contained in [ ] are optional. Expressions $e^{*}_{\mid}$ means any number of e expressions separated by $\mid$.

the sorts. One type of constraint is a *stratified (place) sorting* that determines, for a given sort, the particular sorts of the children. An extension to binding bigraphs [6] is possible. This approach uses properties of port-sortings to form link-sortings, and details how to construct a sub-sorting relation (which we do not explore here).

Birkedal *et al.* [5] give a categorical approach to specifying sorts based on selecting an appropriate functor that determines correctly shaped bigraphs within the category of all bigraphs. While useful, it is not clear how to practically specify the shape category. This approach is reminiscent of type-graphs [24] where the well sortedness of a graph can be checked by verifying an appropriate mapping from instance to type graph exists. This type of approach does not work well for bigraphs as bigraphs are categorical arrows, not objects, and so all structural checks need to be through defining valid decompositions[2].

3 A Pattern Language of Sorts

To allow sorts to be useful for practical bigraph modelling scenarios, we must move sort definitions from textual descriptions and into a formal language amenable to parsing and analysis.

For usability, we introduce a new language similar to algebraic data type definitions seen in programming languages but with some major differences:

1. There is support for specifying not just record (product) like structures, but also explicit support for typing of links.
2. Most languages order constructor parameters, *e.g.* $s(x, y) \neq s(y, x)$. We treat $s(x, y)$ and $s(y, x)$ as equivalent as bigraphs do not have fixed child orders.

[2] Similar issues are seen in [1].

The grammar of our sort definition language is given in Fig. 2. We use strings starting with lowercase to identify sorts, and strings starting with an uppercase to identify entity constructors. We use fonts, *e.g.* A, to distinguish entities from sorts that we denote with font $\mathtt{s}$. $\mathbb{1}$ is the unit sort. We also have a sort $\mathbb{0}$, the empty sort, that is used to mark entities as atomic. A user never specifies $\mathbb{0}$ directly so it is not included in the grammar.

Sorts may be defined either constructorless or with entity constructors. Constructorless sorts, *e.g.* $\mathtt{sort\ s}$ is useful for defining link sortings since we may wish to give a link port a different sort than the entity the port is on. Constructor-based sorts have the form $\mathtt{sort\ t} = \mathsf{A}\ \mathtt{s} \mid \mathsf{B}\ \mathtt{w}$ that specifies entities A and B have sort $\mathtt{t}$, and the patterns for their children: in this case a single $\mathtt{s}$ sorted child for A and $\mathtt{w}$ sorted child for B. Each entity belongs to a single sort, and it is an open question if this could be weakened, *e.g.* to allow sub-typing relationships.

Sort patterns can be combined in two ways: $\mathtt{s} \times \mathtt{t}$ creates a product sort, *i.e.* specifying the need for (exactly) two child bigraphs of sort $\mathtt{s}$ **and** $\mathtt{t}$; while $\mathtt{s} + \mathtt{t}$ is a sum sort, *i.e.* specifying a child must have sort $\mathtt{s}$ **or** $\mathtt{t}$. For product sorts, there is no need for $\mathtt{s}$ and $\mathtt{t}$ to differ, *e.g.* $\mathsf{B}\ \mathtt{n} \times \mathtt{n}$ specifies that a B entity must have **exactly two** $\mathtt{n}$ sorted children, and in this way allows cardinality constraints to be encoded. Finally, $\mathtt{s}^*$ is the (infinite) product sort of $\mathtt{s}$, *e.g.* $\mathsf{A}\ \mathtt{s}^*$ allows A to have any number of $\mathtt{s}$ sorted children—including none. Sorts may be defined recursively, *e.g.* $\mathtt{sort\ s} = \mathsf{A}\ \mathtt{s}^*$ is well defined.

Atomic entities, that never have children are specified as the constructor name and no pattern. When analysing sorts, we treat, for example, A as $\mathsf{A}\ \mathbb{0}$.

Link patterns use the notation, *e.g.* $\mathtt{sort\ s} = \mathsf{A}\{\mathtt{t} \to \mathtt{w} + \mathtt{y}\}$, that specifies A has a[3] port with port-sort $\mathtt{t}$ that is part of a link that additionally has either a $\mathtt{w}$ or $\mathtt{y}$ sorted port on it. In any link pattern $\mathtt{t} \to \mathtt{w}$, we call $\mathtt{t}$ the *domain sort* and $\mathtt{w}$ the *range sort*. Domain sorts must be a base sort. For entities with arity greater than one, we simply list all port sort patterns.

Formally, the grammar defines sorting signatures in the form $\Sigma = (\Theta, \mathcal{K}, \mathcal{L})$, with Θ mapping constructors to sorts, $\mathcal{K}$ constructors to sort patterns, and $\mathcal{L}$ constructors to link patterns.

Example 1 (Sorting Petri nets). Our model has sorts:

$$\begin{array}{ll}
\mathtt{sort\ m} = \mathsf{Tok} & \mathtt{sort\ l} = \mathsf{In}\{\mathtt{l} \to \mathtt{o} \times \mathtt{l}^*\} \mid \mathsf{Out}\{\mathtt{l} \to \mathtt{i} \times \mathtt{l}^*\} \\
\mathtt{sort\ i} & \mathtt{sort\ t} = \mathsf{Tran}\{\mathtt{i} \to \mathtt{l}^*, \mathtt{o} \to \mathtt{l}^*\} \\
\mathtt{sort\ o} & \mathtt{sort\ p} = \mathsf{Place}\ \mathtt{m}^* \times \mathtt{l}^*
\end{array}$$

Which encodes properties including: 1. Places may contain any number of tokens, including none, 2. All controls except Place are atomic, 3. Places only connect to transitions (via explicit links In and Out that encode a direction), and transitions only to places, *i.e.* we cannot connect two transitions. 4. Transitions may connect to/from nowhere (sources/sinks) as $\mathtt{l}^*$ allows closed links. 5. Places may have no links, *e.g.* we allow disconnected places, but could be strengthened using $\mathtt{l} \times \mathtt{l}^*$ to force at least a single incoming or outgoing link.

[3] Ports are not ordered in bigraphs.

$$[\![\mathbb{0}]\!] = \emptyset \tag{1}$$
$$[\![\mathbb{1}]\!] = \{(\!|\,|\!)\} \tag{2}$$
$$[\![\mathtt{s}]\!] = \{(\!|\mathtt{s}|\!)\} \text{ where } \mathtt{s} \text{ is a base sort} \tag{3}$$
$$[\![\mathtt{s} + \mathtt{t}]\!] = [\![\mathtt{s}]\!] \cup [\![\mathtt{t}]\!] \tag{4}$$
$$[\![\mathtt{s} \times \mathtt{t}]\!] = \{A \uplus B \mid A \in [\![\mathtt{s}]\!], B \in [\![\mathtt{t}]\!]\} \tag{5}$$
$$[\![\mathtt{s}^*]\!] = [\![\mathbb{1}]\!] \cup [\![\mathtt{s}]\!] \cup [\![\mathtt{s} \times \mathtt{s}]\!] \cup [\![\mathtt{s} \times \mathtt{s} \times \mathtt{s}]\!] \cup \ldots \tag{6}$$

Fig. 3: Mapping between sort patterns and set-of-multisets representation.

3.1 Compatibility of Patterns

To ensure a bigraph is well sorted we need to determine when two sorts are *compatible*. For example, we want `s` compatible with `s` and also `s + v` *etc.* We now develop a theory that determines when two sorts are compatible. We have chosen this notion of compatibility as it captures a wide range of examples (Section 5), but it may be that a different notion of compatibility is useful in future, *e.g.* one that allows subsorts to be specified.

Multiple patterns can represent the same sorting constraints. For example $\mathsf{A}\ \mathtt{w} \times \mathtt{y}$ and $\mathsf{A}\ \mathtt{y} \times \mathtt{w}$ both specify that some entity A has one `w` sorted and one `y` sorted child, *i.e.* there is some notion of commutativity in the patterns.

Inspired by Fowler *et al.* [12, 11], who need to reason over a similar pattern language—in this case not to specify structure constraints, but instead to specify possible elements within an asynchronous mailbox—we map our pattern language into a sets-of-multisets structure that gives us the required axioms/structural equivalences for free. This mapping is given in Fig. 3. We use $(\!|s, a, a|\!)$ to denote a multiset containing elements s, a and a, with the usual multiset functions defined, *e.g.* $(\!|a|\!) \uplus (\!|a, b|\!) = (\!|a, a, b|\!)$.

Using this mapping, the pattern $\mathtt{s} \times (\mathtt{t} + \mathtt{w})$ is encoded as:

$$\begin{aligned}
[\![\mathtt{s} \times (\mathtt{t} + \mathtt{w})]\!] &= \{A \uplus B \mid A \in [\![\mathtt{s}]\!], B \in [\![\mathtt{t} + \mathtt{w}]\!]\} \\
&= \{A \uplus B \mid A \in \{(\!|\mathtt{s}|\!)\}, B \in ([\![\mathtt{t}]\!] \cup [\![\mathtt{w}]\!])\} \\
&= \{A \uplus B \mid A \in \{(\!|\mathtt{s}|\!)\}, B \in \{(\!|\mathtt{t}|\!), (\!|\mathtt{w}|\!)\}\} \\
&= \{(\!|\mathtt{s}|\!) \uplus (\!|\mathtt{t}|\!), (\!|\mathtt{s}|\!) \uplus (\!|\mathtt{w}|\!)\} \\
&= \{(\!|\mathtt{s}, \mathtt{t}|\!), (\!|\mathtt{s}, \mathtt{w}|\!)\}
\end{aligned}$$

Sort $\mathbb{1}$ is a unit for $\times$ but not for $+$. This allows expressing when entities *may* have no children (or links might be closed), *e.g.* $\mathtt{s} + \mathbb{1}$ is either empty or of sort `s`. Sort $\mathbb{0}$ is a unit for $+$ and an absorbing element for $\times$. $+$ is idempotent, $+$ and $\times$ are commutative, and $\times$ distributes over $+$.

Given the set-of-multisets encoding of patterns, checking pattern compatibility, which we denote $\bowtie$, consists mainly of checking for a non-empty intersection of

the encoded patterns. We have:

$$\mathtt{s} \bowtie \mathtt{t} \;\; \textbf{iif} \;\; [\![\mathtt{s}]\!] = [\![\mathtt{t}]\!] \textbf{ or } [\![\mathtt{s}]\!] \cap [\![\mathtt{t}]\!] \neq \{\}$$

The use of non-empty intersection handles the special case of $\mathbb{0}$ as sort $\mathbb{0}$ is never compatible with anything other than itself (handled by the equality case), and we use this to denote unsortable elements. $\bowtie$ is reflexive and symmetric, but not transitive[4], so is not an equivalence relation. For example, we can check $(\mathtt{s} \times \mathtt{t}) \bowtie (\mathtt{s}\ (\mathtt{s} + \mathtt{t}))$:

$$[\![\mathtt{s} \times \mathtt{t}]\!] = \{(\!|\mathtt{s}, \mathtt{t}|\!)\} \qquad\qquad [\![\mathtt{s} \times (\mathtt{s} + \mathtt{t})]\!] = \{(\!|\mathtt{s}, \mathtt{s}|\!), (\!|\mathtt{s}, \mathtt{t}|\!)\}$$

Which are compatible given the intersection $(\!|\mathtt{s}, \mathtt{t}|\!)$.

Useful compatibilities include $\mathbb{1} \bowtie \alpha^*$ for any sorting pattern α (by definition above), *e.g.* it is explicitly zero or more entities. A sort $\alpha \times \alpha^*$ forces at least one α.

4 Sorting Abstract Bigraphs

We work with abstract bigraphs, where vertices of the graph structure do not have specific identifiers only entity types. Working with the set of primitive bigraphs—from which all other bigraphs can be built—, we show how the sorts change through composition, tensor and closure/renaming of links. As every bigraph is a combination of these primitives, we can sort any bigraph. These bigraphs are well-sorted-by-design, *i.e.* it is impossible to build a bigraph that does not correspond to the (user-defined) sorting scheme.

Bigraph tensor product $\otimes$ does not commute, and any region movement is through an explicit symmetry operator γ. As we want pattern *under* a single entity to commute—*e.g.* $\mathsf{A}\ \mathtt{s} \times \mathtt{t}$ and $\mathsf{A}\ \mathtt{t} \times \mathtt{s}$ should specify the same pattern—we need an additional product type, $\Diamond$, that specifies disjoint patterns that cannot be swapped (unless under an explicit symmetry operator). These only ever appear in the decomposition of a bigraph and a user never writes these directly.

We extend sort pattern compatibility to support $\Diamond$ as follows:

$$(\mathtt{s}_0 \Diamond \mathtt{s}_1 \Diamond \dots) \bowtie (\mathtt{t}_0 \Diamond \mathtt{t}_1 \Diamond \dots) \quad \textbf{iff} \quad \mathtt{s}_0 \bowtie \mathtt{t}_0, \mathtt{s}_1 \bowtie \mathtt{t}_1, \dots$$

That is, we apply pattern compatibility component-wise. $\mathbb{0}$ is a unit for $\Diamond$ and for ease of presentation we assume $\mathbb{0}$ elements are always cancelled out.

Given a set of user-defined sorts Σ (via the pattern language introduced in Section 3) we can now determine the sort of any bigraph. To achieve this, we extend the bigraph interface definition to also track the sort pattern. For a bigraph B mapping n sites, to m regions, and inner names X to outer names Y we have a (pattern) sorted signature:

$$B : \langle n : \mathtt{s}_0 \Diamond \dots \Diamond \mathtt{s}_{n-1}, X : \Gamma \rangle \to \langle m : \mathtt{t}_0 \Diamond \dots \Diamond \mathtt{t}_{m-1}, Y : \Gamma' \rangle$$

[4] For example, $\mathtt{s} \bowtie (\mathtt{s} + \mathtt{t}), (\mathtt{s} + \mathtt{t}) \bowtie (\mathtt{t} + \mathtt{w})$, but $\mathtt{s}$ is not compatible with $(\mathtt{t} + \mathtt{w})$.

which has a parameter of n sites with sorts compatible with pattern $\mathtt{s}_0 \lozenge \ldots \lozenge \mathtt{s}_{n-1}$, and requires a context of m regions compatible with $\mathtt{t}_0 \lozenge \ldots \lozenge \mathtt{t}_{m-1}$.

For links, we create a *sorting context* Γ that stores the link patterns for names in X, *e.g.* $\Gamma = [x : \mathtt{s} \to \mathtt{t}]$. The ordering of names in Γ does not matter and this can be treated like a set[5]. A bigraph can produce a different context Γ' for the set of names Y, which may be smaller, *i.e.* if B closes/substitutes a name. We say two sorting contexts Γ and Δ are sort compatible $\Gamma \bowtie \Delta$ when:

$$X = \{x_0, \ldots, x_{n-1}\} = \operatorname{dom}\Gamma = \operatorname{dom}\Delta \qquad \text{and} \qquad \Gamma(x_i) \bowtie \Delta(x_i)$$

with $0 \leq i < n$ and treating link patterns as $\lozenge$ products:

$$(\mathtt{s} \to \mathtt{t}) \bowtie (\mathtt{s}' \to \mathtt{t}') \quad \textbf{iif} \quad (\mathtt{s} \lozenge \mathtt{t}) \bowtie (\mathtt{s}' \lozenge \mathtt{t}')$$

That is, two contexts are compatible if they have the same names, and the names agree on the sorts (up to sort compatibility).

To make it easier to work with sorted interfaces, we introduce the following notation:

Definition 1 (Sorted Interface). *$\langle\!\langle \mathtt{s}, \Gamma \rangle\!\rangle$ denotes an interface with place sorting $\mathtt{s}$ and link sorting Γ. As sorts determine the* shape *of a bigraph, it is possible to recover a bigraph interface from a sorted interface. For example, $\langle\!\langle \mathtt{v} \lozenge \mathtt{w}, [a : \mathtt{s} \to \mathtt{t}] \rangle\!\rangle$ must have bigraph interface $\langle 2, \{a\} \rangle$.*

Notation. When only describing place or link sorts specifically we sometimes use the reduced notation, *e.g.* $\langle\!\langle \mathtt{s} \lozenge \mathtt{t} \rangle\!\rangle$, or $\langle\!\langle [\mathtt{s} \to \mathtt{t}] \rangle\!\rangle$ when the other components are trivial (single sorted place $\mathbb{0}$, or $\Gamma = [\,]$). Similarly, $\langle\!\langle\, \rangle\!\rangle$ is a shorthand for $\langle\!\langle \mathbb{0}, [\,] \rangle\!\rangle$. We write $A \sharp B$ to indicate sets $A \cap B = \emptyset$. We define iterated operators as follows:

$$\Box_{i<n} \mathtt{s}_i = \mathtt{s}_0 \,\Box\, \cdots \,\Box\, \mathtt{s}_{n-1}$$

with $\Box \in \{\times, \lozenge, +\}$. Bound n is dropped when clear from the context. A sorting context in the form $[x_0 : \mathtt{s}_0 \to \mathtt{t}_0, \ldots, x_{n-1} : \mathtt{s}_{n-1} \to \mathtt{t}_{n-1}]$ is indicated by $[x_i : \mathtt{s}_i \to \mathtt{t}_i]_{i<n}$. Similarly, we write $[\mathtt{s}_i \to \mathtt{t}_i]_{i<n}$ to denote n link patterns. Throughout this section, we use $\Gamma, \Delta, \Theta, \ldots$ and $\alpha, \tau, \mu, \ldots$ to range over sorting contexts and families of sort patterns, *e.g.* variables that can later be unified with a particular sort pattern, respectively.

4.1 Elementary Place Graphs and Constructors

We introduce primitive (sorted) building blocks for bigraphs and then show how any bigraphs can be created from the combination of these.

Bigraph $1 : \langle 0, \emptyset \rangle \to \langle 1, \emptyset \rangle$ represents the *empty* bigraph consisting of a single region. It has sorted equivalent

$$\frac{}{1 : \langle\!\langle\, \rangle\!\rangle \to \langle\!\langle \mathbb{1} \rangle\!\rangle}\ [\textsc{One}]$$

[5] We use the $[\,]$ notation to distinguish between name sets and sorting contexts.

where $\mathbb{1}$ is our unit sort.

For any interface, we can form the identity bigraph $\mathsf{id}_{\langle n,X\rangle} : \langle n, X\rangle \to \langle n, X\rangle$ that consists only of regions/sites and maps from names to themselves. Identity bigraphs have the property (by definition) that they do not change the sorts of their inputs. We sort them with:

$$\frac{\alpha = \Diamond_{i<n}\alpha_i \qquad \mathrm{dom}\,\Gamma = X \qquad \Gamma(x_i \in X) = \tau_i \to \mu_i}{\mathsf{id}_{\langle n,X\rangle} : \langle\!\langle \alpha, \Gamma\rangle\!\rangle \to \langle\!\langle \alpha, \Gamma\rangle\!\rangle} \text{ [Id]}$$

Bigraph $merge_n : \langle n, \emptyset\rangle \to \langle 1, \emptyset\rangle$ places n sites into a single region and is often used before a composition, *e.g.* to allow all children to fall into one site. For sorts, $merge_n$ has the ability to convert the product-of-patterns $\Diamond$ into the sort product $\times$:

$$\frac{}{merge_n : \langle\!\langle \alpha_0 \Diamond \cdots \Diamond \alpha_{n-1}\rangle\!\rangle \to \langle\!\langle \alpha_0 \times \cdots \times \alpha_{n-1}\rangle\!\rangle} \text{ [Merge]}$$

As with identity bigraphs, this is a family of merge operators (with α_i any sort pattern) based on the specific sorts/size of the merge.

Symmetries are sorted as follows:

$$\frac{}{\gamma_{m,n} : \langle\!\langle (\Diamond_{i<m}\alpha_i)\,\Diamond(\Diamond_{j<n}\alpha_j)\rangle\!\rangle \to \langle\!\langle (\Diamond_{j<n}\alpha_j)\,\Diamond(\Diamond_{i<m}\alpha_i)\rangle\!\rangle} \text{ [Sym]}$$

Finally, for each user defined entity and a sorting signature $\Sigma = (\Theta, \mathcal{K}, \mathcal{L})$, we sort ions—which are bigraphs consisting of a single entity, *i.e.* an entity constructor—$\mathsf{K}_X : \langle 1, \emptyset\rangle \to \langle 1, X = \{x_0, \ldots, x_{n-1}\}\rangle$ with the following rule:

$$\frac{\Theta(\mathsf{K}) = \mathtt{w} \qquad \mathcal{K}(\mathsf{K}) = \mathtt{v} \qquad \mathcal{L}(\mathsf{K}) = [\mathtt{s}_i \to \mathtt{t}_i]_{i<n}}{\mathsf{K}_X : \langle\!\langle \mathtt{v}\rangle\!\rangle \to \langle\!\langle \mathtt{w}, [x_i : \mathtt{s}_i \to \mathtt{t}_i]_{i<n}\rangle\!\rangle} \text{ [Ion]}$$

For example, given signature $\mathtt{sort\ s} = \mathsf{A}\{\mathtt{t} \to \mathtt{v}\}\ \mathtt{n} \mid \mathsf{B}\ \mathtt{m} + \mathtt{n}$ we get ions:

$$\mathsf{A}_x : \langle\!\langle \mathtt{n}\rangle\!\rangle \to \langle\!\langle \mathtt{s}, [x : \mathtt{t} \to \mathtt{v}]\rangle\!\rangle \quad \text{and} \quad \mathsf{B} : \langle\!\langle \mathtt{m} + \mathtt{n}\rangle\!\rangle \to \langle\!\langle \mathtt{s}\rangle\!\rangle$$

4.2 Combining Place Graphs

With the elementary primitives in place, we show how to combine bigraphs into larger bigraphs. The bigraph theory is based on a symmetric monoidal category and so the main operators are tensor $\otimes$—that places two bigraphs side-by-side—and composition $\circ$—that puts regions into sites, and joins common names.

The sorted tensor introduces products-of-patterns (these are separate regions so should not commute):

$$\frac{F : \langle\!\langle \alpha, \Gamma\rangle\!\rangle \to \langle\!\langle \tau, \Gamma'\rangle\!\rangle \qquad G : \langle\!\langle \varphi, \Delta\rangle\!\rangle \to \langle\!\langle \mu, \Delta'\rangle\!\rangle \qquad \begin{array}{c}\mathrm{dom}\,\Gamma \sharp \mathrm{dom}\,\Delta \\ \mathrm{dom}\,\Gamma' \sharp \mathrm{dom}\,\Delta'\end{array}}{G \otimes F : \langle\!\langle \alpha \Diamond \varphi, \Gamma \cup \Delta\rangle\!\rangle \to \langle\!\langle \tau \Diamond \mu, \Gamma' \cup \Delta'\rangle\!\rangle} \text{ [Tens]}$$

Here names must be disjoint. The tensor rule only builds larger sort types but does not do any sort checking.

Composition, which places a bigraph into the parameter of another bigraph, performs sort compatibility checking during composition as follows:

$$\frac{F : \langle\!\langle \alpha, \Gamma \rangle\!\rangle \to \langle\!\langle \varphi, \Gamma' \rangle\!\rangle \qquad G : \langle\!\langle \tau, \Delta \rangle\!\rangle \to \langle\!\langle \mu, \Delta' \rangle\!\rangle \qquad \begin{array}{c} \varphi \bowtie \tau \\ \Gamma' \bowtie \Delta \end{array}}{G \circ F : \langle\!\langle \alpha, \Gamma \rangle\!\rangle \to \langle\!\langle \mu, \Delta' \rangle\!\rangle} \text{ [Comp]}$$

The $\bowtie$ constraints enforce that composition only occurs when the sort patterns of outer and inner face (both link and place sorts) are compatible.

4.3 Elementary Link Graphs

For links, we have two elementary bigraphs: closure, which stops a name from moving to the context, and substitution which combines (a set of) inner names into a single outer name. Practically, this is how we join links, *e.g.* in Fig. 4, a substitution joins two links (from A and B) to a single link with outer name y.

To sort the outer names, we need to determine if we can produce a valid *extension* of a sort pattern, *i.e.* the minimum we would need to include in the context to make a sort compatibility $\bowtie$ hold. We make use of the sets-of-multisets encoding to check when this is the case. We use $t \preceq s$ when there exists *any* element in $[\![t]\!]$ that is a sub-multiset of an element in $[\![s]\!]$. This gives, for example, $s \preceq s \times t$ as $[\![s \times t]\!] = \{(\!|s, t|\!)\}, [\![s]\!] = \{(\!|s|\!)\}$ and $(\!|s|\!) \subseteq (\!|s, t|\!)$. $\preceq$ does not imply pattern compatibility, although this may hold in some cases.

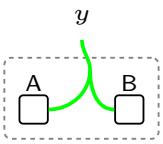


Fig. 4: Example bigraph $B = (merge_2 \otimes y/\{x_0, x_1\}) \circ (\mathsf{A}_{x_0} \otimes \mathsf{B}_{x_1})$.

To make it clear how to use $\preceq$ in practice, we show how to sort a link by example, and then generalise to a sorting rule for substitutions.

Example 2. Consider bigraph B in Fig. 4, and the following ($\mathtt{s}, \mathtt{t}$ and $\mathtt{w}$ are constructorless sorts):

$$\mathtt{sort\ a} \ = \mathsf{A}\{\mathtt{s} \to \mathtt{s} \times \mathtt{t}\} \mid \mathsf{B}\{\mathtt{s} \to \mathtt{s} \times (\mathtt{t} + \mathtt{w})\}$$

We need to check there is a possible sorting of ports of A, B, and y that makes the link well-sorted. The link has known domain sorts $\mathtt{p} = \mathtt{s} \times \mathtt{s}$ from taking the domain sort from each link pattern. We then check each link constraint is met by the domain sorts in turn. For $\mathsf{A}\{\mathtt{s} \to \mathtt{s} \times \mathtt{t}\}$ we form the pattern $\mathtt{m}_0 = \mathtt{s} \times \mathtt{s} \times \mathtt{t}$,

i.e. we introduce the domain sort into the range sort pattern. We then check this constraint is a valid context extension of the domain sorts, *i.e.* checking $\mathtt{p} \preceq \mathtt{m}_0$. This holds as $(\!|\mathtt{s},\mathtt{s}|\!) \subseteq (\!|\mathtt{s},\mathtt{s},\mathtt{t}|\!)$ as required. For $\mathsf{B}\{\mathtt{s} \to \mathtt{s} \times (\mathtt{t} + \mathtt{w})\}$ we have $\mathtt{m}_1 = \mathtt{s} \times \mathtt{s} \times (\mathtt{t} + \mathtt{w})$ and $\mathtt{p} \preceq \mathtt{m}_1$ because $(\!|\mathtt{s},\mathtt{s}|\!) \subseteq (\!|\mathtt{s},\mathtt{s},\mathtt{t}|\!)$ as required. Note that $(\!|\mathtt{s},\mathtt{s},\mathtt{w}|\!) \in [\![\mathtt{s} \times \mathtt{s} \times (\mathtt{t} + \mathtt{w})]\!]$, but we only need one sub-multiset to hold. At this point we know that there are valid sorts on the links and the final piece is to determine a suitable sort for y (the extended context). We find this through:

$$[\![\bigcap_{i<2} [\![\mathtt{m}_i - \mathtt{p}]\!]]\!]^{-1} \quad \text{with} \quad [\![\mathtt{s} - \mathtt{t}]\!] = \{A \setminus B \mid A \in [\![\mathtt{s}]\!], B \in [\![\mathtt{t}]\!]\} \tag{7}$$

where $[\![\,]\!]^{-1}$ is the inverse of the encoding defined in Fig. 3. As $\mathtt{s} - \mathtt{t}$ always returns a single set of multisets, the intersection of n sets of multisets is a single (or empty) set of multisets, and there is a image for any single set of multisets in Fig. 3, the inverse is always defined. While $[\![\,]\!]$ is not injective in general, *e.g.* $[\![[\![\mathbb{1} \times \mathbb{1}]\!]]\!]^{-1} = \mathbb{1}$ it captures the intended semantics (sort compatibility), *i.e.* we want to treat $\mathbb{1} \bowtie (\mathbb{1} \times \mathbb{1})$.

In our case:

$$[\![\mathtt{m}_0]\!] = [\![\mathtt{s} \times \mathtt{s} \times \mathtt{t}]\!] = \{(\!|s,s,t|\!)\}$$
$$[\![\mathtt{m}_0 - \mathtt{p}]\!] = \{(\!|s,s,t|\!) \setminus (\!|s,s|\!)\} = \{(\!|t|\!)\}$$

$$[\![\mathtt{m}_1]\!] = [\![\mathtt{s} \times \mathtt{s} \times (\mathtt{t} + \mathtt{w})]\!] = \{(\!|s,s,t|\!), (\!|s,s,w|\!)\}$$
$$[\![\mathtt{m}_1 - \mathtt{p}]\!] = \{(\!|s,s,t|\!) \setminus (\!|s,s|\!), (\!|s,s,w|\!) \setminus (\!|s,s|\!)\} = \{(\!|t|\!), (\!|w|\!)\}$$

$$\{(\!|t|\!)\} \cap \{(\!|t|\!), (\!|w|\!)\} = \{(\!|t|\!)\}$$
$$[\![\{(\!|t|\!)\}]\!]^{-1} = \mathtt{t}$$

We now know that the port sorts can be extended to make a valid link, and that the extension needs to be $y : \mathtt{s} \times \mathtt{s} \to \mathtt{t}$ *i.e.* the link taking our port patterns to $\mathtt{t}$. The substitution $y/\{x_0, x_1\}$ can be safely made and we only need to consider the name y in future.

Formally, for a set of names $X = \{x_0, \ldots, x_{n-1}\}$, with $n > 0$, and a name y, substitutions $y/X : \langle 0, X\rangle \to \langle 0, \{y\}\rangle$ are sorted by the following rule:

$$\frac{\alpha = \times_{i<n}\alpha_i \qquad \forall_{i<n}(\alpha \preceq \alpha_i \times \tau_i) \qquad \mu = \mathcal{S}(\alpha, [x_i : \alpha_i \to \tau_i]_{i<n})}{y/X : \langle\!\langle [x_i : \alpha_i \to \tau_i]_{i<n} \rangle\!\rangle \to \langle\!\langle [y : \alpha \to \mu] \rangle\!\rangle} \text{ [RNAME]}$$

which states all existing ports on the links need to be valid on all link constraints, and computes the remainder by generalising the construction given in Eq. (7) to define $\mathcal{S}$

$$\mathcal{S}(\alpha, [x_i : \alpha_i \to \tau_i]_{i<n}) = [\![\bigcap_{i<n} ((\alpha_i \times \tau_i) - \alpha)]\!]^{-1}$$

In practice we defer applying $\mathcal{S}(\,)$ until we have concrete sorts. When $X = \emptyset$, *i.e.* the empty substitution $y : \langle 0, \emptyset\rangle \to \langle 0, \{y\}\rangle$, we apply the rule below:

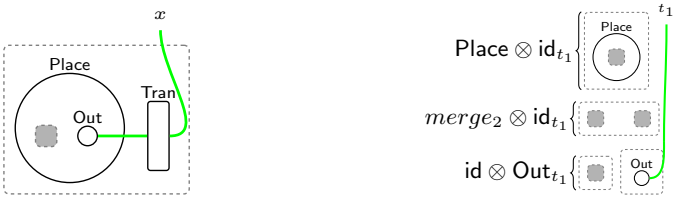


Fig. 5: (a) Worked example: simple Petri net model; (b) Partial algebraic decomposition (terms compose vertically).

$$\frac{\mu \text{ fresh sort}}{y : \langle\!\langle\rangle\!\rangle \to \langle\!\langle [y : \mathbb{1} \to \mu] \rangle\!\rangle} \text{ [New]}$$

Intuitively, as there are no ports in the link, no constraints need to be propagated up to the context, as represented by the fresh sort in the premise.

Finally, for closures $/x$ we ensure the range sort of x maps to a sort compatible with $\mathbb{1}$ at which point nothing else needs to be added to the link.

$$\frac{\tau \bowtie \mathbb{1}}{/x : \langle\!\langle [x : \alpha \to \tau] \rangle\!\rangle \to \langle\!\langle\rangle\!\rangle} \text{ [Close]}$$

4.4 Worked Example

We show how to sort the simple Petri net model shown in Fig. 5 with respect to the sorts in Example 1. An algebraic form for the example (partially shown in Fig. 5b) is:

$$P = \overbrace{(\mathsf{Place} \otimes \mathsf{id}_{t_1})}^{P'} \circ \underbrace{\overbrace{(merge_2 \otimes \mathsf{id}_{t_1})}^{M} \circ \overbrace{(\mathsf{id} \otimes \mathsf{Out}_{t_1})}^{O}}_{P''}$$

$$B = \overbrace{(/y \otimes \mathsf{id}_{1,x})}^{B'} \circ \underbrace{\overbrace{(y/\{t_1, t_2\} \otimes \mathsf{id}_x \otimes merge_2)}^{M'} \circ \overbrace{(P \otimes \mathsf{Tran}_{t_2,x})}^{T}}_{B''}$$

We check the sortings beginning with bigraph P. Simple applications of rules Tens, Ion, Merge, and Id give us the sorts for the elementary subterms

$$P' : \langle\!\langle \mathtt{m}^* \times \mathtt{1}^*, \Theta \rangle\!\rangle \to \langle\!\langle \mathtt{p}, \Theta \rangle\!\rangle \qquad M : \langle\!\langle \tau \lozenge \lambda, \Gamma \rangle\!\rangle \to \langle\!\langle \tau \times \lambda, \Gamma \rangle\!\rangle$$
$$O : \langle\!\langle \alpha \rangle\!\rangle \to \langle\!\langle \alpha \lozenge \mathtt{1}, \Delta \rangle\!\rangle$$

where $\Theta = [t_1 : \gamma]$, $\Gamma = [t_1 : \beta]$, $\Delta = [t_1 : \mathtt{d}]$, and $\mathtt{d} = \mathtt{1} \to \mathtt{i} \times \mathtt{1}^*$. Term P'' is sorted by applying rule Comp

$$\frac{O : \langle\!\langle \alpha \rangle\!\rangle \to \langle\!\langle \alpha \lozenge \mathtt{1}, \Delta \rangle\!\rangle \qquad M : \langle\!\langle \tau \lozenge \lambda, \Gamma \rangle\!\rangle \to \langle\!\langle \tau \times \lambda, \Gamma \rangle\!\rangle \qquad \begin{array}{c} \tau \lozenge \lambda \bowtie \alpha \lozenge \mathtt{1} \\ \tau \bowtie \alpha \quad \lambda \bowtie \mathtt{1} \\ \Gamma \bowtie \Delta \\ \implies \lambda = \mathtt{1} \quad \beta = \mathtt{d} \end{array}}{P'' : \langle\!\langle \alpha \rangle\!\rangle \to \langle\!\langle \alpha \times \mathtt{1}, \Delta \rangle\!\rangle}$$

where we highlight the unifiers required for sort inference. Families of sorts are always compatible so no substitution is needed to satisfy $\tau \bowtie \alpha$. By further applying Comp we can sort P

$$\frac{P'' : \langle\!\langle \alpha \rangle\!\rangle \to \langle\!\langle \alpha \times \mathtt{l}, \Delta \rangle\!\rangle \qquad P' : \langle\!\langle \mathtt{m}^* \times \mathtt{l}^*, \Theta \rangle\!\rangle \to \langle\!\langle \mathtt{p}, \Theta \rangle\!\rangle \qquad \begin{array}{c} \mathtt{m}^* \times \mathtt{l}^* \bowtie \alpha \times \mathtt{l} \\ \Theta \bowtie \Delta \\ \Longrightarrow\ \alpha = \mathtt{m}^* \times \mathtt{l}^* \\ \gamma = \mathtt{d} \end{array}}{P : \langle\!\langle \mathtt{m}^* \times \mathtt{l}^* \rangle\!\rangle \to \langle\!\langle \mathtt{p}, \Delta \rangle\!\rangle}$$

Here we have removed many fresh sorting variables since the ion context, *e.g.* the sort of Place, forces a sort for α (inferred using a similar process to Eq. (7) but for place graphs, *i.e.* intuitively since we have $\mathtt{l}^*$, removing an $\mathtt{l}$ still leaves an $\mathtt{l}^*$) and likewise for the identity links on t_1.

Extending to add the transition entity through an application of Tens, we obtain $T : \langle\!\langle \mathtt{m}^* \times \mathtt{l}^* \rangle\!\rangle \to \langle\!\langle \mathtt{p} \lozenge \mathtt{t}, \Delta' \rangle\!\rangle$ where $\Delta' = [t_1 : \mathtt{d}, t_2 : \mathtt{e}, x : \mathtt{f}]$, $\mathtt{e} = \mathtt{i} \to \mathtt{l}^*$, and $\mathtt{f} = \mathtt{o} \to \mathtt{l}^*$.

At B'' we are composing into a substitution and so we follow the link sorting procedure as before:

$$\frac{\dfrac{\dfrac{\gamma' \preceq \alpha_1 \times \tau_1 \quad \gamma' \preceq \alpha_2 \times \tau_2 \qquad \mu = \mathcal{S}(\gamma', \Gamma'')}{y/\{t_1, t_2\} : \langle\!\langle \Gamma'' \rangle\!\rangle \to \langle\!\langle y : \gamma' \to \mu \rangle\!\rangle} \ \vdots}{M' : \langle\!\langle \phi \lozenge \psi, \Gamma' \rangle\!\rangle \to \langle\!\langle \phi \times \psi, \Theta' \rangle\!\rangle} \qquad \begin{array}{c} \phi \lozenge \psi \bowtie \mathtt{p} \lozenge \mathtt{t} \\ \phi \bowtie \mathtt{p} \quad \psi \bowtie \mathtt{t} \\ \Gamma' \bowtie \Delta' \\ \Longrightarrow\ \phi = \mathtt{p} \quad \psi = \mathtt{t} \\ \alpha_1 = \mathtt{l} \quad \alpha_2 = \mathtt{i} \quad \alpha_3 = \mathtt{f} \\ \tau_1 = \mathtt{i} \times \mathtt{l}^* \quad \tau_2 = \mathtt{l}^* \\ \mu = \mathtt{l}^* \end{array}}{B'' : \langle\!\langle \mathtt{m}^* \times \mathtt{l}^* \rangle\!\rangle \to \langle\!\langle \mathtt{p} \times \mathtt{t}, \Theta'' \rangle\!\rangle}$$

where $\gamma' = \alpha_1 \times \alpha_2$, $\Gamma'' = [t_1 : \alpha_1 \to \tau_1, t_2 : \alpha_2 \to \tau_2]$, $\Gamma' = \Gamma'' \cup [x : \alpha_3]$, $\Theta' = [y : \gamma' \to \mu, x : \alpha_3]$, and $\Theta'' = [y : \mathtt{l} \times \mathtt{i} \to \mathtt{l}^*, x : \mathtt{f}]$.

Finally, we apply rule Close to check y may be closed—which is possible as $\mathtt{l}^* \bowtie \mathbb{1}$—and so the example is well sorted with final sort for B:

$$\frac{\dfrac{\dfrac{\tau \bowtie \mathbb{1}}{/y : \langle\!\langle [y : \alpha \to \tau] \rangle\!\rangle \to \langle\!\langle\, \rangle\!\rangle} \ \vdots}{B' : \langle\!\langle \gamma, \Delta \rangle\!\rangle \to \langle\!\langle \gamma, \Delta' \rangle\!\rangle} \qquad \begin{array}{c} \gamma \bowtie \mathtt{p} \times \mathtt{t} \\ \Delta \bowtie \Theta'' \\ \Longrightarrow\ \gamma = \mathtt{p} \times \mathtt{t} \\ \alpha = \mathtt{l} \times \mathtt{i} \quad \tau = \mathtt{l}^* \quad \beta = \mathtt{f} \end{array}}{B : \langle\!\langle \mathtt{m}^* \times \mathtt{l}^* \rangle\!\rangle \to \langle\!\langle \mathtt{p} \times \mathtt{t}, [x : \mathtt{o} \to \mathtt{l}^*] \rangle\!\rangle}$$

where $\Delta' = [x : \beta]$ and $\Delta = \Delta' \cup [y : \alpha \to \tau]$. Notice this *must* be a top-level bigraph, *i.e.* cannot compose elsewhere, as we have no constructors that accept the sort $\mathtt{p} \times \mathtt{t}$.

5 Different Sorts of Application

Using our new syntax, we show how to encode sorting schemes for a range of existing models in the literature.

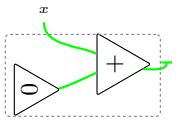


Fig. 6: Arithmetic Nets example: $\bot = 0 + x$

Arithmetic Nets. A model of Arithmetic Nets is given in [19, Chapter 6], and an example is in Fig. 6. These nets wire components to denote arithmetic result flow. We show a reduced model with only Zero and Plus components[6]. While similar to Petri-nets, differ by the lack of component nesting.

Milner assigns these nets sorts `s` (source) and `t` (target) and creates a rule set (in plain text, not amenable to implementation) that: 1. every link has only one source, 2. a *link* has sort `s` if it has a source on it, 3. and closed links always have sort `s`. As we only sort *ports* (and constrain through links) we cannot express the last two constraints directly, nor can we describe a closed link specifically. The essence of these sorts can be maintained by ensuring we never have `s` on both the left and right of a link pattern and only ever as a single sort (*i.e.* never as $\mathtt{s}^*$):

$$\begin{aligned}&\mathtt{sort\ s}\\&\mathtt{sort\ t}\\&\mathtt{sort\ e} = \mathsf{Zero}\{\mathtt{s} \to \mathtt{t}^*\} \mid \mathsf{Plus}\{\mathtt{t} \to \mathtt{s}, \mathtt{t} \to \mathtt{s}, \mathtt{s} \to \mathtt{t}^*\}\end{aligned}$$

Zero entities only have one source (connected to any number of targets), while Plus entities have two targets/inputs (from one source node each), and creates a source/output (connected to any number of future targets).

Small changes in the sorting can specify significantly different nets. For example:

$$\mathtt{sort\ e} = \mathsf{Zero}\{\mathtt{s} \to \mathtt{t}^*\} \mid \mathsf{Plus}\{\mathtt{t} \to \mathtt{s}, \mathtt{t} \to \mathtt{s}, \mathtt{s} \to \mathtt{t} \times \mathtt{t}^*\}$$

Ensures a Plus node always connects somewhere (but allows the source of Zero to be closed).

$$\mathtt{sort\ e} = \mathsf{Zero}\{\mathtt{s} \to \mathtt{t}\} \mid \mathsf{Plus}\{\mathtt{t} \to \mathtt{s}, \mathtt{t} \to \mathtt{s}, \mathtt{s} \to \mathtt{t}\}$$

Makes all links binary (and unclosed), *e.g.* the source of Zero always connects to *exactly* one target port of a Plus.

$$\mathtt{sort\ e} = \mathsf{Zero}\{\mathtt{s} \to \mathtt{t}\} \mid \mathsf{Plus}\{\mathtt{t} \to (\mathbb{1} + \mathtt{s}), \mathtt{t} \to (\mathbb{1} + \mathtt{s}), \mathtt{s} \to \mathtt{t}^*\}$$

Allows some targets/inputs to Plus to be unspecified (closed); perhaps treating them implicitly as 0 internally.

[6] Milner also gives the successor function, and a node that forwards an input to an output.

CCS. Milner gives a model and sorting scheme for CCS in [19]. The main sorting constraint is that *alternations* always contain *processes* and *processes* always contain *alternations*. We express this in our sortings using:

$$\mathtt{sort}\ \mathtt{a} = \mathsf{Send}\{\mathtt{a} \to \mathtt{a}^*\}\ \mathtt{p}^* \mid \mathsf{Get}\{\mathtt{a} \to \mathtt{a}^*\}\ \mathtt{p}^*$$
$$\mathtt{sort}\ \mathtt{p} = \mathsf{Alt}\ \mathtt{a}^*$$

That is, send and get *can* (but do not need to) connect other sends and gets via hyperedges, and they always contain processes. Alternations can contain many alternative processes including the *nil* process (modelled implicitly by the empty region 1).

λ-Calculus. Currently our approach does not support binding bigraphs (where names can have locality constraints), however we can still specify the sorts for a λ-calculus models such as that in [17]. A possible sorting is:

$$\mathtt{sort}\ \mathtt{exp} = \mathsf{Var}\{\mathtt{exp} \to \mathtt{exp}^*\} \mid \mathsf{Lam}\{\mathtt{exp} \to \mathtt{exp}^*\}\ \mathtt{exp} \mid \mathsf{App}\ \mathtt{l} \times \mathtt{r}$$
$$\mathtt{sort}\ \mathtt{l} = \mathsf{Left}\ \mathtt{exp}$$
$$\mathtt{sort}\ \mathtt{r} = \mathsf{Right}\ \mathtt{exp}$$

This encodes the three main components of λ calculus: variables, which connect to all like-named[7] variables/binders, abstractions that bind a (new) name, and function application that includes the important constraint that App must contain exactly one left and one right component: something that is currently difficult to enforce without a sorting scheme.

Virus Spread. Our sortings are not only useful for computational models. A model for the spread of a virus through a network is given in [14] and adapted to bigraphs in [2]. Network nodes have a specific status—safe, attacked, or infected—and connect to other nodes through a nesting of links (similar to the Petri net model) which allows a virus to spread. A possible sorting is:

$$\mathtt{sort}\ \mathtt{n} = \mathsf{Safe}\ \mathtt{l}^* \mid \mathsf{Attacked}\ \mathtt{l}^* \mid \mathsf{Infected}\ \mathtt{l}^*$$
$$\mathtt{sort}\ \mathtt{l} = \mathsf{Link}\{\mathtt{l} \to \mathtt{l}\}$$

Here the status of a node is encoded in the entity type. We allow flexible network configurations through Link entities, but enforce in the sorting that these links are always binary—something that has been difficult to express without sorts.

Formal Results. Full formal analysis is out of scope, but one result of interest is showing how the category of sorted bigraphs relates to the category of unsorted bigraphs. Bigraphical categories have interfaces as objects and bigraphs as morphisms. The goal of sorting is to essentially filter badly formed morphisms.

[7] We do not need an explicit notion of names as links perform the role of binders.

For sorted bigraphs we have objects $\langle\!\langle\, \rangle\!\rangle$ and morphsims defined by the sorting rules and (user defined) sorts. We can define a functor mapping sorted interfaces to unsorted equivalents (there is always a way to recover this as described in Definition 1). Identities are preserved by sorting rule Id (which sorts any interface). Composition is preserved by rule Comp, *i.e.* whenever two sorted bigraphs compose, because they have the correct interface in the non-sorted version (and this is the only requirement), they must also compose there.

6 Conclusions

For practical modelling it is important to be able to restrict the range of bigraphs that can be created, *e.g.* to avoid nesting physical entities within virtual spaces, and utilising sorting schemes is a promising approach. In contrast to existing approaches that lack computational descriptions, we have defined a language for specifying sorts, and shown how, by extending the usual bigraph interfaces to sorted variants, a pathway to practical sorting is possible.

Expressivity and Limitations. While the new sorting approach is flexible, and captures a wide range of practical models, there are some constraints that are difficult to express.

We only define sort patterns for *entities*, and while we can use this to infer the sorts of names/sites/regions we cannot *specify* the sort for a region, *e.g.* we cannot say all regions are `s` sorted. This can lead to cases, such as the Petri net of Section 4.4, where we have a well-sorted bigraph that cannot compose anywhere.

We only constrain the sorts of *direct* children, and we cannot encode constraints such as "there must be a `s` sorted grandchild". This type of constraint sometimes appears in the literature [7]. More generally, we cannot express global constraints, *e.g.* that only one instance of a particular sort exists in an entire model (singleton types). Implementing binding bigraphs as a sorting would require similar expressiveness for both placement and links. One enabler for this in future might be to make use of spatial logics [8, 10].

For links, we can express when a link is *allowed* to be closed but cannot force a link to be closed. We also only have notions of port-sorts and it remains open if this is enough to express existing link-sort constraints (*e.g.* the models in Section 5), and how useful link-sorts are in practice.

Future Work. We plan to implement this approach within the BigraphER tool [22]. We will explore the interplay of sorts with reaction rules (that specify dynamics), in particular showing how to manage instantiation maps (that can manipulate sites during rewriting) so that sorts are preserved. We believe restricting to solid bigraphs [13], which BigraphER already does to handle probabilistic/stochastic bigraphs, is also beneficial for sorting since it ensures the sorts of the left-hand of a rule is unambiguous.

We will also consider how sorts allow generation of random bigraphs for model testing; explore the theoretical connections to existing sorting schemes and

bigraph concepts *e.g.* RPOs; and extend our rules to handle variants of bigraphs including bigraphs with sharing [21] and conditional bigraphs [1].

References

1. Archibald, B., Calder, M., Sevegnani, M.: Conditional bigraphs. In: Gadducci, F., Kehrer, T. (eds.) Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12150, pp. 3–19. Springer (2020). https://doi.org/10.1007/978-3-030-51372-6_1
2. Archibald, B., Calder, M., Sevegnani, M.: Probabilistic bigraphs. Formal Aspects Comput. **34**(2), 1–27 (2022). https://doi.org/10.1145/3545180
3. Bacci, G., Grohmann, D.: On decidability of bigraphical sorting. In: International Workshop on Graph Computation Models (2010)
4. Baez, J.C., Master, J.: Open petri nets. Math. Struct. Comput. Sci. **30**(3), 314–341 (2020). https://doi.org/10.1017/S0960129520000043
5. Birkedal, L., Debois, S., Hildebrandt, T.T.: On the construction of sorted reactive systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5201, pp. 218–232. Springer (2008). https://doi.org/10.1007/978-3-540-85361-9_20
6. Bundgaard, M., Sassone, V.: Typed polyadic pi-calculus in bigraphs. In: Bossi, A., Maher, M.J. (eds.) Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy. pp. 1–12. ACM (2006). https://doi.org/10.1145/1140335.1140336
7. Calder, M., Sevegnani, M.: Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing. Formal Aspects Comput. **26**(3), 537–561 (2014). https://doi.org/10.1007/S00165-012-0270-3
8. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings. pp. 222–235 (2014). https://doi.org/10.1007/978-3-662-44602-7_18
9. Conchúir, S.T.O.: Explicit Substitution and Sorted Bigraphs. Ph.D. thesis, Trinity College, Dublin, Ireland (2009), http://www.tara.tcd.ie/handle/2262/83173
10. Conforti, G., Macedonio, D., Sassone, V.: Static BiLog: a unifying language for spatial structures. Fundam. Informaticae **80**(1-3), 91–110 (2007), http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-06
11. de'Liguoro, U., Padovani, L.: Mailbox types for unordered interactions. In: Millstein, T.D. (ed.) 32nd European Conference on Object-Oriented Programming, ECOOP 2018, July 16-21, 2018, Amsterdam, The Netherlands. LIPIcs, vol. 109, pp. 15:1–15:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPICS.ECOOP.2018.15
12. Fowler, S., Attard, D.P., Sowul, F., Gay, S.J., Trinder, P.: Special delivery: Programming with mailbox types. Proc. ACM Program. Lang. **7**(ICFP), 78–107 (2023). https://doi.org/10.1145/3607832
13. Krivine, J., Milner, R., Troina, A.: Stochastic bigraphs. In: Bauer, A., Mislove, M.W. (eds.) Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2008, Philadelphia, PA, USA, May 22-25, 2008. Electronic Notes in Theoretical Computer Science, vol. 218, pp. 73–96. Elsevier (2008). https://doi.org/10.1016/j.entcs.2008.10.006

14. Kwiatkowska, M.Z., Norman, G., Parker, D., Vigliotti, M.G.: Probabilistic mobile ambients. Theor. Comput. Sci. **410**(12-13), 1272–1303 (2009). https://doi.org/10.1016/J.TCS.2008.12.058
15. Leifer, J.J., Milner, R.: Transition systems, link graphs and petri nets. Math. Struct. Comput. Sci. **16**(6), 989–1047 (2006). https://doi.org/10.1017/S0960129506005664
16. Milner, R.: Bigraphs for petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]. Lecture Notes in Computer Science, vol. 3098, pp. 686–701. Springer (2003). https://doi.org/10.1007/978-3-540-27755-2_19
17. Milner, R.: Local bigraphs and confluence: Two conjectures: (extended abstract). In: Amadio, R.M., Phillips, I. (eds.) Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006. Electronic Notes in Theoretical Computer Science, vol. 175, pp. 65–73. Elsevier (2006). https://doi.org/10.1016/J.ENTCS.2006.07.035
18. Milner, R.: Pure bigraphs: Structure and dynamics. Inf. Comput. **204**(1), 60–122 (2006). https://doi.org/10.1016/j.ic.2005.07.003
19. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press (2009)
20. Peterson, J.L.: Petri nets. ACM Comput. Surv. **9**(3), 223–252 (1977). https://doi.org/10.1145/356698.356702
21. Sevegnani, M., Calder, M.: Bigraphs with sharing. Theor. Comput. Sci. **577**, 43–73 (2015). https://doi.org/10.1016/j.tcs.2015.02.011
22. Sevegnani, M., Calder, M.: BigraphER: Rewriting and analysis engine for bigraphs. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 494–501. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_27
23. Sevegnani, M., Pereira, E.: Towards a bigraphical encoding of actors (June 2014), http://eprints.gla.ac.uk/94772/
24. Taentzer, G., Rensink, A.: Ensuring structural constraints in graph-based models with type inheritance. In: Cerioli, M. (ed.) Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3442, pp. 64–79. Springer (2005). https://doi.org/10.1007/978-3-540-31984-9_6