An Implementation of Bigraph Matching

Arne John Glenstrup
Troels Christoffer Damgaard
Lars Birkedal
Espen Højsgaard
An Implementation of Bigraph Matching

Arne John Glenstrup    Troels Christoffer Damgaard    Lars Birkedal
Espen Højsgaard

IT University of Copenhagen, Denmark

Abstract
We describe a provably sound and complete matching algorithm for bigraphical reactive systems. The algorithm has been implemented in our BPL Tool, a first implementation of bigraphical reactive systems. We describe the tool and present a concrete example of how it can be used to simulate a model of a mobile phone system in a bigraphical representation of the polyadic $\pi$ calculus.

1 Introduction

The theory of bigraphical reactive systems [13] provides a general meta-model for describing and analyzing mobile and distributed ubiquitous systems. Bigraphical reactive systems form a graphical model of computation in which graphs embodying both locality and connectivity can be reconfigured using reaction rules. So far it has been shown how to use the theory for recovering behavioural theories for various process calculi [12, 13, 15] and how to use the theory for modelling context-aware systems [2].

In this paper we describe the core part of our BPL Tool, a first prototype implementation of bigraphical reactive systems, which can be used for experimenting with bigraphical models.

The main challenge of implementing the dynamics of bigraphical reactive systems is the matching problem, that is, to determine for a given bigraph and reaction rule whether and how the reaction rule can be applied to rewrite the bigraph. When studying the matching problem in detail, one finds that it is a surprisingly tricky problem (it is related to the NP-complete graph embedding problem). Therefore we decided early on to study the matching problem quite formally and base our prototype implementation on a provably correct specification. In previous work [1, 9], we gave a sound and complete inductive characterization of the matching problem for bigraphs. Our inductive characterization was based on normal form theorems for binding bigraphs [8].

In the present paper we extend the inductive characterization from graphs to a term representation of bigraphs. A single bigraph can be represented by several structurally congruent bigraph terms. Using an equational theory for bigraph terms [8], we essentially get a non-deterministic matching algorithm operating on bigraph terms. However, such an algorithm will be wildly non-deterministic and we thus provide an alternative, but still provably sound and complete, characterization of matching on terms, which is more suited for mechanically finding matching. In particular, it spells out how and where to make use of structural congruences.

We have implemented the resulting algorithm in our BPL Tool, which we briefly describe in Section 8. We also present an example of a bigraphical reactive system, an encoding of the polyadic $\pi$ calculus, and show how it can be used to simulate a simple model of a mobile phone system.

Bigraphical reactive systems are related to general graph transformation systems; Ehrig et al. [10] provide a recent comprehensive overview of graph transformation systems. In particular, bigraph matching is related to the general graph pattern matching (GPM) problem, so general GPM algorithms might also be applicable to bigraphs [11, 14, 20, 21]. As an alternative to implementing matching for bigraphs, one
could try to formalize bigraphical reactive systems as graph transformation systems and then use an existing implementation of graph transformation systems. Some promising steps in this direction have been taken [19], but they have so far fallen short of capturing precisely all the aspects of binding bigraphs. For a more detailed account of related work, in particular on relations between BRSS, graph transformations, term rewriting and term graph rewriting, see the Thesis of Damgaard [7, Section 6].

The remainder of this paper is organized as follows. In Section 2 we give an informal presentation of bigraphical reactive systems and normalisation techniques needed for the implementation. In Section 3 we recall the graph-based inductive characterization, then in Section 4 we develop a term-based inductive characterization, forming the basis for our implementation of matching. Section 5 explains how we can restrict the kind of inference trees the algorithm needs to consider, without sacrificing completeness; this is then used in Section 6, where we describe how to translate the inference system into a working algorithm. We discuss how to handle nondeterminism in Section 7, and in Section 8 we describe the BPL Tool and present an example use of it. Finally, we conclude and discuss future work in Section 9.

2 Bigraphs and Reactive Systems

In the following, we present bigraphs informally; for a formal definition, see the work by Jensen and Milner [13] and Damgaard and Birkedal [8].

2.1 Concrete Bigraphs

A concrete binding bigraph $G$ consists of a place graph $G^P$ and a link graph $G^L$. The place graph is an ordered list of trees indicating location, with roots $r_0, \ldots, r_n$, nodes $v_0, \ldots, v_k$, and a number of special leaves $s_0, \ldots, s_m$ called sites, while the link graph is a general graph over the node set $v_0, \ldots, v_k$ extended with inner names $x_0, \ldots, x_l$, and equipped with hyper edges, indicating connectivity.

We usually illustrate the place graph by nesting nodes, as shown in the upper part of Figure 1 (ignore for now the interfaces denoted by “$\cdot \rightarrow \cdot$”). A link is a hyper edge of the link graph, either an internal edge $e$ or a name $y$. Links that are names are called open, those that are edges are called closed. Names
and inner names can be *global* or *local*, the latter being located at a specific root or site, respectively. In Figure 1, $y_0$ is located at $r_0$, indicated by a small ring, and $x_0$ and $x_2$ are located at $s_2$, indicated by writing them within the site. Global names like $y_1$ and $y_2$ are drawn anywhere at the top, while global inner names like $x_1$ are drawn anywhere at the bottom. A link, including internal edges like $e_2$ in the figure, can be located with one *binder* (the ring), in which case it is a *bound link*, otherwise it is *free*. However, a bound link must satisfy the *scope rule*, a simple structural requirement that all points (cf. next paragraph) of the link lie within its location (in the place graph), except for the binder itself. This prevents $y_2$ and $e_1$ in the example from being bound.

### 2.2 Controls and Signatures

Every node $v$ has a *control* $K$, indicated by $v : K$, which determines a binding and free arity. In the example of Figure 1, we could have $v_1 : K_i, i = 0, 1, 2, 3$, where arities are given by $K_0 : 1, K_1 : 2, K_2 : 3, K_3 : 1 \rightarrow 2$, using $K : f$ as a shorthand for $K : 0 \rightarrow f$. The arities determine the number of bound and free *ports* of the node, to which bound and free links, respectively, are connected. Ports and inner names are collectively referred to as *points*.

In addition to arity, each control is assigned a *kind*, either atomic, active or passive, and describe nodes according to their control kinds. We require that atomic nodes contain no nodes except sites; any site being a descendant of a passive node is passive, otherwise it is active. If all sites of a bigraph $G$ are active, $G$ is active.

A collection of controls with their associated kinds and arities is referred to as a *signature*.

### 2.3 Abstract Bigraphs

While concrete bigraphs with named nodes and internal edges are the basis of bigraph theory [13], our prime interest is in *abstract bigraphs*, equivalence classes of concrete bigraphs that differ only in the names of nodes and internal edges\(^1\). Abstract bigraphs are illustrated with their node controls (see Figure 14 in Section 8). In what follows, “bigraph” will thus mean “abstract bigraph.”

### 2.4 Interfaces

Every bigraph $G$ has two *interfaces* $I$ and $J$, written $G : I \rightarrow J$, where $I$ is the inner face and $J$ the outer face. An interface is a triple $(m, \bar{X}, X)$, where $m$ is the *width* (the number of sites or roots), $X$ the entire set of local and global names, and $\bar{X}$ indicates the locations of each local name, cf. Figure 1. We let $\epsilon = (0, [], {})$; when $m = 1$ the interface is *prime*, and if all $x \in X$ are located by $\bar{X}$, the interface is *local*. As in the work by Milner [18] we write $G : \rightarrow J$ or $G : I \rightarrow$ for $G : I \rightarrow J$ when we are not concerned about about $I$ or $J$, respectively.

A bigraph $G : I \rightarrow J$ is called *ground*, or an *agent*, if $I = \epsilon$, prime if $I$ is local and $J$ prime, and a *wiring* if $m = n = 0$, where $m$ and $n$ are the widths of $I$ and $J$, respectively. For $I = (m, \bar{X}, X)$, bigraph $id_I : I \rightarrow I$ consists of $m$ roots, each root $r_i$ containing just one site $s_i$, and a link graph linking each inner name $x \in X$ to name $x$.

### 2.5 Discrete and Regular Bigraphs

We say that a bigraph is *discrete* iff every free link is a name and has exactly one point. The virtue of discrete bigraphs is that any connectivity by internal edges must be bound, and node ports can be accessed individually by the names of the outer face. Further, a bigraph is *name-discrete* iff it is discrete and every

\(^1\)Formally, we also disregard *idle* edges: edges not connected to anything.

---

3
bound link is either an edge, or (if it is a name) has exactly one point. Note that name-discrete implies discrete.

A bigraph is regular if, for all nodes v and sites i, j, k with \( i \leq j \leq k \), if i and k are descendants of v, then j is also a descendant of v. Further, for roots \( r_f \) and \( r_f' \), and all sites i and j where i is a descendant of \( r_f \) and j of \( r_f' \), if \( i \leq j \) then \( i' \leq j' \). The bigraphs in the figures are all regular, the permutation in Table 1 is not. The virtue of regular bigraphs is that permutations can be avoided when composing them from basic bigraphs.

### 2.6 Product and Composition

For bigraphs \( G_1 \) and \( G_2 \) that share no names or inner names, we can make the tensor product \( G_1 \otimes G_2 \) by juxtaposing their place graphs, constructing the union of their link graphs, and increasing the indexes of sites in \( G_2 \) by the number of sites of \( G_1 \). We write \( \otimes \) for the iterated tensor \( G_0 \otimes \cdots \otimes G_{n-1} \), which, in case \( n = 0 \), is id. 

The parallel product \( G_1 \parallel G_2 \) is like the tensor product, except global names can be shared: if y is shared, all points of y in \( G_1 \) and \( G_2 \) become the points of y in \( G_1 \parallel G_2 \).

The prime product \( G_1 \mid G_2 \) is like the parallel product, except the result has just one root (also when \( G_1 \) and \( G_2 \) are wirings), produced by merging any roots of \( G_1 \) and \( G_2 \) into one.

We can compose bigraphs \( G_2 : I \rightarrow F \) and \( G_1 : F \rightarrow J \), yielding bigraph \( G_1 \circ G_2 : I \rightarrow J \), by plugging the sites of \( G_1 \) with the roots of \( G_2 \), eliminating both, and connecting names of \( G_2 \) with inner names of \( G_1 \). In the following, we will omit the ‘◦’ and simply write \( G_1G_2 \) for composition, letting it bind tighter than tensor product.

### 2.7 Notation, Basic Bigraphs, and Abstraction

In the sequel, we will use the following notation: \( \psi \) denotes union of sets required to be disjoint; we write \( \{ \bar{Y} \} \) for \( Y_0 \psi \cdots Y_{n-1} \) when \( \bar{Y} = Y_0 \ldots Y_{n-1} \), and similarly \( \{ \bar{y} \} \) for \( \{ y_0 , \ldots , y_{n-1} \} \). For interfaces, we write \( n \) to mean \( \langle n , [0, \ldots , 0] , \emptyset \rangle \), \( X \) to mean \( \langle 0 , \emptyset , X \rangle \), \( \langle X \rangle \) to mean \( \langle 1 , \emptyset , X \rangle \) and \( \rho \) to mean \( \langle 1 , [X] , X \rangle \).

Any bigraph can be constructed by applying composition, tensor product and abstraction to identities (on all interfaces) and a set of basic bigraphs, shown in Table 1 [8]. For permutations, when used in any context, \( \pi_XG \) or \( G\pi_X \), \( \bar{X} \) is given entirely by the interface of \( G \); in these cases we simply write \( \pi_X \) as \( \pi \).

Given a prime \( P \), the abstraction operation localises a subset of its outer names. Note that the scope rule is necessarily respected since the inner face of a prime \( P \) is required to be local, so all points of \( P \) are located within its root. The abstraction operator is denoted by \( \cdot \) and reaches as far right as possible.

For a renaming \( \alpha : X \rightarrow Y \), we write \( \cdot \alpha^{-1} \) to mean \( (\alpha \otimes \text{id}_X)^{-1} X' \), and when \( \sigma : U \rightarrow Y \), we let \( \bar{\sigma} = (Y)(\sigma \otimes \text{id}_X)^{-1} U \). We write substitutions \( \bar{y}/[\emptyset , \ldots , \emptyset ] : e \rightarrow Y \) as \( Y \).

Note that \( [\emptyset ]/[] = [\emptyset ]/\emptyset = \pi_0 = \text{id}_e \) and \( \text{merge}_1 = [\emptyset ]/\emptyset = \pi_1 = \text{id}_1 \), where \( \pi_1 \) is the nameless permutation of width 1.

### 2.8 Bigraphical Reactive Systems

Bigraphs in themselves model two essential parts of context: locality and connectivity. To model also dynamics, we introduce bigraphical reactive systems (BRS) as a collection of rules. Each rule \( R \stackrel{\rho}{\rightarrow} R' \) consists of a regular redex \( R : I \rightarrow J \), a reactum \( R' : I' \rightarrow J \), and an instantiation \( \rho \), mapping each site of \( R' \) to a site of \( R \), and mapping local names in \( I' \) to those of \( I \), as illustrated in Figure 2. Interfaces \( I = \langle m , \bar{X} , X \rangle \) and \( I' = \langle m' , \bar{X}' , X' \rangle \) must be local, and are related by \( X'_i = X_{\rho(i)} \), where \( \rho \) must be a bijection between \( X'_i \) and \( X_{\rho(i)} \). We illustrate \( \rho \) by \( i := i' \), whenever \( \rho(i) = j \neq i \), or, alternatively, by listing \( [\rho(0), \ldots , \rho(m' - 1)] \). Given an instantiation \( \rho \) and a discrete bigraph \( d = d_0 \otimes \cdots \otimes d_k \) with prime \( d_i \)'s, we let \( \rho(d) = d_{\rho(0)} \otimes \cdots \otimes d_{\rho(k)} \), allowing copying, discarding and reordering parts of \( d \).
### Table 1: Basic bigraphs, the abstraction operation, and variables ranging over bigraphs.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge</td>
<td>(\text{merge}_n : n \rightarrow 1) (\text{merge}_3 = )</td>
</tr>
<tr>
<td>Concretion</td>
<td>(\langle X \rangle : (X) \rightarrow \langle X \rangle)</td>
</tr>
<tr>
<td>Abstraction</td>
<td>((Y)P : I \rightarrow ({Y}, Z \uplus Y))</td>
</tr>
<tr>
<td>Substitution</td>
<td>(\bar{y}/\bar{X} : X \rightarrow Y)</td>
</tr>
<tr>
<td>Renaming</td>
<td>(\bar{y}/\bar{X} : X \rightarrow Y)</td>
</tr>
<tr>
<td>Closure</td>
<td>(/X : X \rightarrow {})</td>
</tr>
<tr>
<td>Wiring</td>
<td>((\text{id} \otimes /Z)\sigma : X \rightarrow Y)</td>
</tr>
<tr>
<td>Ion</td>
<td>(K_{\bar{y}(\bar{X})} : ({\bar{X}}) \rightarrow ({\bar{y}}))</td>
</tr>
<tr>
<td>Permutation</td>
<td>(\pi_X : (i_1 \mapsto \ldots) : (m, \bar{X}, X) \rightarrow (m, \pi(\bar{X}), X))</td>
</tr>
</tbody>
</table>

**Figure 2:** A reaction rule
Given an agent \( a \), a match of redex \( R \) is a decomposition \( a = C(id_Z \otimes R)d \), with active context \( C \) and discrete parameter \( d \) with its global names \( Z \). Dynamics is achieved by transforming \( a \) into a new agent \( a' = C(id_Z \otimes R')d' \), where \( d' = \rho(d) \), cf. Figure 3. This definition of a match is as given by Jensen and Milner [13], except that we here also require \( R \) to be regular. This restriction to regular redexes \( R \) simplifies the inductive characterization of matching without limiting the set of possible reactions, as sites in \( R \) and \( R' \) can be renumbered to render \( R \) regular.

### 2.9 Bigraph Terms and Normal Forms

Expressing bigraphs as terms composed by product, composition and abstraction over basic bigraph terms, Damgaard and Birkedal [8] showed that bigraphs can be expressed on normal forms uniquely up to certain permutations and renamings. Further, they showed equivalence of term and bigraph equality, which will allow us in Section 4 to base our implementation on terms rather than graphs.

In this work, we use the normal forms shown in Figure 4, enabling us to express regular bigraphs simply by removing the permutations. These normal forms are unique up to permutation of \( S_i \)'s and renaming of names not visible on the interfaces.

\[
\begin{align*}
M &::= (id_Z \otimes K_{\bar{y}(\bar{X})})N &\text{molecule} \\
S &::= \top | M &\text{singular top-level node} \\
G &::= (id_Y \otimes \text{merge}_n)(\bigotimes_i S_i)\pi &\text{global discrete prime} \\
N &::= (X)G &\text{name-discrete prime} \\
P, Q &::= (id_Z \otimes \hat{\sigma})N &\text{discrete prime} \\
D &::= \alpha \otimes (\bigotimes_i P_i)\pi &\text{discrete bigraph} \\
B &::= (\omega \otimes id_{\bar{X}})D &\text{binding bigraph}
\end{align*}
\]

Figure 4: Normal forms for binding bigraphs

### 2.10 Normalising

For normalising an arbitrary bigraph \( t \), we define a normalisation relation \( t \Downarrow_B t' \) for bigraph terms (details are given in Figure 22 of Appendix A.1), with the following property:

**Proposition 1** For any bigraph terms \( t, t' \), if \( t \) represents a bigraph \( b \) and \( t \Downarrow_B t' \), then \( t' \) represents \( b \) as well, and is on \( B \)-normal form given in Figure 4.
The relation is straightforward, recursively normalising subterms and recombining the results; for tensor product, the rule stated is

\[
\begin{align*}
t_i \downarrow B & \quad (\omega \otimes \text{id}_{\vec{Y}_i})D_i \\
\omega &= \bigotimes_{j \in \mathbb{N}} \alpha_j \quad \alpha = \bigotimes_{j \in \mathbb{N}} \alpha_j \quad \text{id}_{\vec{Y}_i} = \bigotimes_{j \in \mathbb{N}} \text{id}_{\vec{Y}_j} \\
P &= \bigotimes_{j \in \mathbb{N}} p_j^i \\
D_i &\equiv \alpha \otimes (\bigotimes_{j \in \mathbb{N}} p_j^i) \pi_i : I_i \to (n_i, \vec{Y}_i, Y_i) \\
\bigotimes_{i \in \mathbb{N}} t_i \downarrow B \quad (\omega \otimes \text{id}_{\vec{Y}_i})D
\end{align*}
\]

We find that the expression \( \bigotimes_{j \in \mathbb{N}} p_j^i \) in general will lead to name clashes, because we can only assume that outer, not inner names, of the \( \alpha_j \)'s are disjoint.

One solution could be to rename names on \( p_j^i \)'s outer face in the Bten rule. However, as Bten is applied recursively at each level of tensor product, this would lead to multiple renamings of the same names, causing inefficiency. Instead, we precede normalisation by a renaming phase described in the following; it will prevent name clashes in normalisation.

### 2.11 Renaming

While renaming names used in a term might look trivial at first sight, it is in fact not entirely straightforward. First, inner and outer names of a term must not be renamed, or we would be representing a different bigraph. Second, we cannot even require of a renamed term that all internal names are unique, as a normalised subterm can contain several instances of the same name, due to the use of \( \text{id}_Y \) in the normal form.

Thus, we need to identify a more refined notion of internal horizontal uniqueness, where a name can be reused vertically in link compositions, but not horizontally in tensor products. To this end, given a term \( t \), we conceptually replace all occurrences of \( /X \) by \( e_1/x_1 \otimes \cdots \otimes e_n/x_n \), and \( K_{\vec{Y}/X} \) by \( K_{\vec{Y}/\vec{X}} \), in effect naming uniquely each closed link. We then define a function \( \text{linknames} \), mapping terms to link names (details are given in Figure 23 of Appendix A.2). Using this function we define a predicate normalisable, which identifies terms whose tensor products and compositions do not produce subterms with name clashes, and is preserved by normalisation (details are given in Figure 24 of Appendix A.2):

**Proposition 2** For any bigraph term \( t \), if normalisable(\( t \)), there exists a \( t' \) such that \( t \downarrow B t' \) and normalisable(\( t' \)).

For the actual renaming, we define inductively a renaming judgment \( U \vdash _B t \downarrow B t', \beta : V \), where \( U \) is a set of used names and \( \alpha \) renames \( t \)'s inner names to those of \( t' \), while \( \beta \) renames \( t \)'s outer names to those of \( t' \) and \( V \) extends \( U \) with names used in \( t' \) (details are given in Figure 25 of Appendix A.2).

We can show that renaming preserves the bigraph, and enables normalisation:

**Proposition 3** Given a term \( t \) representing a bigraph \( b : \langle m, \vec{X}, X \rangle \rightarrow \langle n, \vec{Y}, Y \rangle \), we can derive \( X \cup Y \vdash _B t \downarrow B t', \beta : V \) for some \( t' \), \( \beta : V \), and set \( t' = \left( (\beta^{\text{glob}})^{-1} \otimes (\beta^{\text{loc}})^{-1} \right) t'' \); then \( t' \) represents \( b \), and normalisable(\( t' \)).

### 2.12 Regularising

As a regular bigraph can be expressed as a term containing permutations, we must define regularising to represent it as a permutation-free term. This is done by splitting the permutations in the \( D \)- and \( G \)-normal forms, recursively pushing them into the subterms where they reorder the tensor product of \( S_i \)'s.

While \( D \)'s permutation \( \pi \) must be a tensor product of \( \pi_j \)'s—otherwise the bigraph would not be regular—\( G \)'s permutation, on the other hand, need not be so. However, as the bigraph is regular, it must be possible to split it into a major permutation \( \pi_X \) and \( n \) minor permutations \( \pi_Y \), based on the local
inner faces, \( \vec{X} \), of the \( S_i \)'s. Then \( \vec{X} \) is elided by permuting the \( S_i \)'s, and each \( \vec{X} \) permutation is handled recursively in its \( S_i \) (details are given in Figure 26 of Appendix A.3).

We can show that regularisation is correct:

**Proposition 4** Given a term \( t \) representing a regular bigraph \( b \), we can infer \( t \rightarrow t' \), for some \( t' \) where \( t' \) contains no nontrivial permutations, and \( t' \) represents \( b \).

### 2.13 Summary

A detailed illustration of the entire reaction cycle including the preceding transformation technologies can be seen in Figure 5.

![Figure 5: Details of the reaction cycle](image-url)

### 3 Inferring Matches Using a Graph Representation

In this section we recap matching inference using a graph representation as developed in [9]; this representation is the basis for correctness proofs.

For simplicity, we will first consider just place graphs to explain the basic idea behind matching inference.

#### 3.1 Matching place graphs

A place graph match is captured by a matching sentence:

**Definition 5 (Matching Sentence for Place Graphs)** A matching sentence for place graphs is a 4-tuple of bigraphs \( a, R \rightarrow C, d, \) all are regular except \( C \), with \( a \) and \( d \) ground. A sentence is valid iff \( a = CRd \).

We infer place graph matching sentences using the inference system given in Figure 6. Traversing an inference tree bottom-up, the agent is decomposed, while constructing the context, using the \( \text{ION}, \text{MERGE} \) and \( \text{PAR} \) rules. The \( \text{PERM} \) rule permutes redex parts to align tensor factors with corresponding agent factors.

At the point in the agent where a redex root should match, leaving a site in the context, the \( \text{SWITCH} \) rule is applied, switching the roles of the context and redex. This allows the remaining rules to be reused (above the switch rule) for checking that the redex matches the agent. When a site in the redex is reached, whatever is left of the agent should become (a part of) the parameter—this is captured by the \( \text{PRIME-AXIOM} \) rule.

For a match with a redex \( R : m \rightarrow n \) consisting of \( n \) nontrivial (i.e., non-identity) primes, the inference tree will contain \( m \) applications of \( \text{PRIME-AXIOM} \) and \( n \) applications of \( \text{SWITCH} \). Further, between any
leaf and the root of the inference tree, SWITCH will be applied at most once. The structure of a matching inference tree will thus generally be as illustrated in Figure 7; rules applied above SWITCH match agent and redex structure, while rules applied below match agent and context structure.

### 3.2 Matching binding bigraphs

Turning now to consider binding bigraphs, we extend the matching sentences to cater for links:

**Definition 6 (Matching Sentence for Binding Bigraphs)** A (binding bigraph) matching sentence is a 7-tuple of bigraphs: $\omega_a, \omega_R, \omega_C \vdash a, R \rightarrow C, d$, where $a, R, C$ and $d$ are discrete with local inner faces, all regular except $C$, with $a$ and $d$ ground. It is valid iff $(\text{id} \otimes \omega_a)a = (\text{id} \otimes \omega_C)(\text{id}_Z \otimes V \otimes C)(\text{id}_Z \otimes (\text{id} \otimes \omega_R)R)d$.

This definition separates the wirings, leaving local wiring in $a$, $R$, $C$ and $d$, while keeping global wiring of agent, redex and context in $\omega_a$, $\omega_R$ and $\omega_C$, respectively; this is possible for any agent, redex and context [9]. The validity property shows how a valid matching sentence relates to a match, as illustrated in Figure 8.

To reach a system for inferring valid matching sentences for binding bigraphs, we simply augment the place graph rules with wirings as shown in Figure 9, and add three rules for dealing with purely wiring constructs, shown in Figure 10. A detailed explanation of the rules is available in the literature [9], along with proofs of soundness and completeness of the inference system.

### 4 From Graph Matching to Term Matching

In this section we transform the graph based inductive characterisation of matching to be based on a term representation in such a way that correctness and completeness is preserved.
\[
\begin{align*}
\text{PRIME-AXIOM} & \quad \begin{array}{l}
\sigma : W \oplus U \rightarrow \beta : Z \rightarrow U \quad \alpha : V \rightarrow W \quad \tau : X \rightarrow V \quad p : \langle X \uplus Z \rangle \\
\sigma(\beta \otimes \alpha \tau), \text{id}_e, \sigma \vdash p, \text{id}_V \rightarrow \alpha, \beta(\beta \otimes \tau)(X) p
\end{array} \\
\text{ION} & \quad \begin{array}{l}
\omega_a, \omega_R, \omega_C \vdash \langle (\bar{v} / \bar{X}) \otimes \text{id}_Y \rangle p, R \leftarrow \langle (\bar{v} / \bar{Z}) \otimes \text{id}_W \rangle P.d \quad \alpha = \bar{y} / \bar{u} \quad \sigma : \{\bar{y}\} \rightarrow \langle y / \bar{X}, Y, X \uplus Z \rangle \\
\sigma \mid \omega_a, \omega_R, \sigma \alpha \mid \omega_C \vdash \langle (\bar{v} / \bar{X}) \otimes \text{id}_Y \rangle p, R \leftarrow \langle (\bar{v} / \bar{Z}) \otimes \text{id}_W \rangle P.d
\end{array} \\
\text{SWITCH} & \quad \begin{array}{l}
\omega_a, \text{id}_e, \omega_C(\sigma \otimes \omega_R \otimes \text{id}_Z) \vdash p, \text{id} \rightarrow \langle \bigoplus (\bar{v} / \bar{X}) \rangle \omega \vdash \langle \bigoplus (\bar{v} / \bar{X}) \rangle \omega \vdash a, R \leftarrow C.d
\end{array} \\
\text{PAR} & \quad \begin{array}{l}
\omega_a, \omega_R, \omega_C \mid \omega \vdash a, R \leftarrow C.d \quad \omega_a, \omega_S, \omega_P \mid \omega \vdash b, S \leftarrow D,e
\end{array} \\
\text{PERM} & \quad \begin{array}{l}
\omega_a, \omega_R, \omega_C \vdash a, \otimes \pi P \rightarrow (\pi \otimes \text{id}) d \\
\omega_a, \omega_R, \omega_C \vdash a, \otimes \pi B \leftarrow C\pi, d
\end{array} \\
\text{MERGE} & \quad \begin{array}{l}
\omega_a, \omega_R, \omega_C \vdash a, R \leftarrow C.d
\end{array}
\end{align*}
\]

Figure 9: Place graph rules (shaded) augmented for deriving binding bigraph matches

\[
\begin{align*}
\text{WIRING-AXIOM} & \quad \begin{array}{l}
y, X, y / X \vdash \text{id}_e, \text{id}_e \rightarrow \text{id}_e, \text{id}_e
\end{array} \\
\text{ABSTR} & \quad \begin{array}{l}
\sigma_a \otimes \omega_a, \omega_R, \sigma_C \otimes \omega_C \vdash p, R \rightarrow P, d \quad \sigma_a : Z \rightarrow W \quad p : \langle Z \uplus Y \rangle \quad \sigma_C : U \rightarrow W \quad P : \rightarrow \langle U \uplus X \rangle \\
\sigma_a, \omega_R, \omega_C \vdash (\sigma_a \otimes \text{id}_Y)(Z)p, R \rightarrow (\sigma_C \otimes \text{id}_X)(U)P.d
\end{array} \\
\text{CLOSE} & \quad \begin{array}{l}
\omega_a, \omega_R, \sigma_C : d, a, R \leftarrow C.d
\end{array}
\end{align*}
\]

Figure 10: Added inference rules for deriving binding bigraph matches
While the graph representation of matching sentences is useful for constructing a relatively simple inference system amenable to correctness proofs, it is not sufficient for an implementation based on syntax, that is, bigraph terms. One bigraph can be represented by several different bigraph terms that are structurally congruent by the axiom rules: $a = a \otimes \text{id}_0 = \text{merge}_1 a$, $a \otimes (b \otimes c) = (a \otimes b) \otimes c$ and $\text{merge}(a \otimes b) = \text{merge}(b \otimes a)$. If, for instance, we were to match agent $a = \text{merge}((K \otimes \text{id}_0) \otimes M)$ with redex $R = K$, we would first need to apply the axioms to achieve $R = \text{merge}((R \otimes \text{id}_0) \otimes a)$ before being able to apply the \text{MERGE} and \text{PAR} rules.

In the following, we recast the matching sentences to be tuples of 3 wirings and 4 bigraph terms $\omega_a, \omega_R, \omega_C \vdash a, R \rightsquigarrow C, d$, with the same restrictions and validity as before, interpreting the terms as the bigraphs they represent. Given this, adding just this one rule would be sufficient to achieve completeness of the inference system:

$$
\text{STRUCT} \quad a \equiv a' \quad R \equiv R' \quad C \equiv C' \quad h \equiv h' \quad \omega^a, \omega^R, \omega^C \vdash a', R' \rightsquigarrow C', h'
\quad \quad \omega^a, \omega^R, \omega^C \vdash a, R \rightsquigarrow C, h
$$

The \text{STRUCT} rule says that we can apply structural congruence to rewrite any term $a, R, C$ or $h$ to a term denoting the same bigraph. With the help of the equational theory for determining bigraph isomorphism on the term level [8], we have essentially a nondeterministic algorithm for matching bigraph terms—implementable in say, Prolog. A brief glance at the equational theory, shows us, though, that the associative and commutative properties of the basic operators of the language would yield a wildly non-deterministic inference system, since we would need to apply structural congruence between every step to infer a match. This is reminiscent of the problems in implementing rewriting logic, that is, term rewriting modulo a set of static equivalences [5, 6, 16]. Consequently, we abandon the fully general \text{STRUCT} rule. For the purposes of stating the completeness theorem below, we shall need to refer to sentences derived from the ruleset for bigraphs (i.e., from section 3.2) recast to terms with the help of the \text{STRUCT} rule above. We shall write such sentences $\omega^a, \omega^R, \omega^C \vdash a, R \rightsquigarrow C, h$ for wirings $\omega^a, \omega^R, \omega^C$ and terms $a, R, C$ and $h$.

**Definition 7** For wirings $\omega^a, \omega^R, \omega^C$ and terms $a, R, C$ and $h$, sentences $\omega^a, \omega^R, \omega^C \vdash a, R \rightsquigarrow C, h$ range over sentences derived from the rules of Figure 10—reading $a, R, C$ and $h$ as terms—extended with the \text{STRUCT} rule.

5 Normal Inferences

Next, we look at how to restrict the term based inductive characterisation of matching as an enabling step for designing an algorithm. We define normal inferences, limiting the set of inferences we need to consider.

Normal inferences is a class of inferences that are complete in the sense that all valid matching sentences can be inferred, but suitably restricted, such that inferences can be built mechanically. In particular, normal inference definitions for term matching spell out how and where to apply structural congruence. As a main trick, we utilize a variant of the normal forms proven complete for binding bigraphs (cf. Section 2.9), lending us a set of uniform representations of classes of bigraphs based directly on terms for bigraphs; we define normal inferences that require each inference to start by rewriting the term to be on normal form.

Before giving the format for normal inferences, we incorporate structural congruence axioms into \text{PRODUCT} and \text{MERGE} rules. We derive rules for iterated tensor product and permutations under merge, arriving at the inference system shown in Figure 11. In this inference system, the terms in the conclusion of every rule except \text{DNF} is in some normal form as given by Figure 4, where $e$ is a discrete prime ($p$) or global discrete prime ($g$). An expression $[t]^G$ means term $t$ expressed on $G$-normal form—for
Figure 11: Inference rules for binding bigraph terms
instance, \([^n\alpha]_G^C\) means \((\text{id}_Y \otimes \text{merge}_i)(\bigotimes_{i=1}^n \alpha)\)—and similarly for the remaining normal forms. The expression \(\rho(n,m)\) denotes the set of \(n\)-\(m\)-partitions. An \(n\)-\(m\)-partition \(\rho\) is a partition of \(\{0,\ldots,n-1\}\) into \(m\) (possibly empty) subsets, and for \(i < m\), \(\rho_i\) is the \(i\)th subset. Given a metavariable \(\mathcal{X}\), \(\mathcal{F}\) ranges over iterated tensor products of \(\mathcal{X}\)’es. As indicated by the superscript, rules \(\text{PER}^\phi\), \(\text{PAR}_n^\phi\) and \(\text{PAR}_\epsilon^\phi\) can be used either on discrete primes \(p\) and \(P\) or global discrete primes \(g\) and \(G\).

The main differences from the preceding inference system is that we have replaced the binary \(\text{PAR}\) rule by two iterative \(\text{PAR}\) rules, \(\text{PAR}_n^\phi\) and \(\text{PAR}_\epsilon^\phi\), and specialised the \(\text{MERGE}\) rule into a rule, \(\text{MER}\), that makes the partitioning of children in an agent node explicit. The \(\text{PAR}_\epsilon^\phi\) rule splits up an iterated tensor product into a number of products matching agent factors, while \(\text{PAR}_n^\phi\) performs the actual inductive inference on each of the factors. (Note, by the way, that \(\text{PAR}_\epsilon^\phi\) and \(\text{MER}_\epsilon^\phi\) correspond just to particular instances of the \(\text{STRUCT}\)-rule, that we abandoned above.)

Furthermore, note that the usage of the previous \(\text{WIRING-AXIOM}\)-rule for introducing idle linkage has been inlined to a side-condition on a slightly generalized \(\text{PAR}\)-rule (i.e., the \(\text{PAR}_n^\phi\)-rule). The \(\sigma^\phi\) in that rule allows us to introduce idle linkage in redex and agent, and link them in context; as previously allowed by the \(\text{WIRING-AXIOM}\)-rule. Hence, \(\text{PAR}_\epsilon^\phi\) also serves as an axiom, introducing 0-ary products of \(\text{id}_\epsilon\)'s on \(G\)- and \(P\)-normal forms.

While this inference system is more explicit about partitioning tensor products (in the \(\text{STRUCT}\)-rule, that we abandoned above.),

Theorem 9 (Normal inferences are sound and complete) For wirings \(\omega^A, \omega^R, \omega^C\) and terms \(a, R, C, d\), we can infer \(\omega^A, \omega^R, \omega^C \vdash a, R \rightarrow_S C, d\) iff we can infer \(\omega^A, \omega^R, \omega^C \vdash a, R \rightarrow C, d\) using a normal inference.
Proof. (Sketch) By induction over the structure of the derivation of the sentence $\omega^a, \omega^R, \omega^C \vdash a, R \rightsquigarrow_S C, d$. We case on the last rule used to conclude this sentence. By the induction hypothesis (IH), we can conclude a normal derivation of the sentence used for concluding $\omega^a, \omega^R, \omega^C \vdash a, R \rightsquigarrow_S C, d$.

**STRUCT:** By IH, we can construct a normal derivation of $\omega^a, \omega^R, \omega^C \vdash a, R' \rightsquigarrow_C C', d'$, with $a = a'$, $R' = R$, $C' = C$ and $d' = d$. This normal derivation can be used directly to conclude also $\omega^a, \omega^R, \omega^C \vdash a', R' \rightsquigarrow_C C', d'$.

**PRIME-AXIOM:** We produce the needed normal inference by starting with an application of $PAX$, which introduces the needed prime bifigraphs and wiring—that is, each term being equal up to structural congruence to the sentence concluded with $PRIME$. Now we proceed to build the needed normal inference by a building first a $D_P$ and then a $D_B$-inference. All steps add only term structure to match a particular normal form, while not changing the denotation of the terms.

**ION:** By IH, we can construct a normal derivation of $\omega_a, \omega_R, \omega_C \vdash ((\vec{v})/(\vec{X}) \otimes id_V)p, R \rightsquigarrow ((\vec{v})/(\vec{Z}) \otimes id_w)P, d$. For this case, we have to unroll that normal derivation up across the whole $D_B$ production except for the last $ABS$-step, concluding with a $PAR_1^P$ step (since we know $p$ and $P$ are prime). We now have a $D_B$ normal inference with an added $ABS$-step, which we can use for concluding an $ION$-step introducing our needed ion. Referring to the grammar in Figure 12, we see that this produces a $D_G$-inference, which we have to lead through two series of $PAR$-$PER$-$MER$ steps (and one $ABS$-step), to produce a full normal inference.

**SWX:** This case needs a little extra care. First, we point out two properties of normal derivations: (i) any $D_G$ and $D_P$ inference without $SWX$ is also a $D_G$ or $D_P$ inference, respectively; and, (ii) any sentence, $\omega_a, \omega_R, \omega_C \vdash a, id \rightsquigarrow C, h$ has a normal derivation with no $SWX$-steps. Both are easily verified.

Now, by the IH, we can construct a normal derivation of a sentence $\omega_a, \omega_R, \omega_C \vdash (\sigma \otimes \omega_R \otimes id_x) \vdash p, id \rightsquigarrow P, d$ for global $P$. By property (ii), we can assume that this normal derivation does not contain any applications of $SWX$. We unroll this normal derivation up across the whole $D_B$ production. This leaves us with a $D_B$-type normal derivation, which by property (i), we can use also as a $D_B$ derivation. Hence, we can apply $SWX$ to obtain a $D_G$ derivation. We proceed to build first a $D_P$ type inference, and then a $D_B$ type inference, in particular applying again $ABS$ to introduce local linkage in $p$.

**PAR:** By IH, we can construct normal derivations of $\omega_a, \omega_R, \omega_C \parallel \omega \vdash a, R \rightsquigarrow C, d$ and $\omega_a, \omega_R, \omega_B \parallel \omega \vdash b, S \rightsquigarrow D, e$. Each of these normal derivations we can unroll up to the last application of $PAR_1^P$ $D_i$ and $\delta_j$, applied for concluding these $PAR_1^P$ steps. To construct the required normal inference we simply let instead a single $PAR_1^P$ step utilize all of the normal inferences $D_i$ and $\delta_j$.

**PERM:** By IH, we can construct a normal derivation of $\omega^a, \omega^R, \omega^C \vdash a, \omega^V \parallel (\pi \otimes id_x) \vdash d$. Unrolling this normal derivation up through the applications of $DNF$, $CLO$, and $PER^P$, we can edit the $PER^P$-step to also move the permutation $\pi$ to the context.

**MERGE:** By IH, we can construct a normal derivation of $\omega_a, \omega_R, \omega_C \vdash a, R \rightsquigarrow C, d$ for global $a$ and $C$. We unroll this derivation up across the $D_B$ production to obtain $n$ $D_B$-derivations (for $a$ and $C$ of width $n$). We may consider these as $D_G$-derivations, also. We combine these in a single application of $PAR_1^P$, and, after a $PAR_1^P$ and a $PER$-step, we apply $MER$ to merge the roots as required by the case. We conclude by adding term structure to the terms of this $D_B$-inference as required by the normal form and lead it through the steps to produce a $D_B$-derivation.

**WIRING-AXIOM:** As sketched in the text above, introduction of idle names is now handled by $PAR_1^P$. For this case, we simply start with a $PAR_1^P$-step and proceed through the grammar for $D_B$ to produce a normal inference as needed.

**ABSTR:** By IH, we can construct a normal derivation of $\sigma_a \otimes \omega_a, \omega_R, \sigma_C \otimes \omega_C \vdash p, R \rightsquigarrow P, d$. We unroll this normal derivation up across the entire $D_B$-inference to obtain a $D_B$ type inference. (We know there is only one $D_B$-inference, as $p$ and $P$ are prime.) We construct the required $D_B$ inference by starting with a modified $ABS$-step, where we introduce the required abstractions and local substitutions.

**CLOSE:** By IH, we can construct a normal inference for a sentence with only substitutions (i.e., with no closed links). We simply unroll this normal inference up across the $CLO$-step, and instead, to produce
the needed normal inference, close the needed names in a new CLO-step. □

Normal inferences are sufficiently restricted such that we can base our prototype implementation on mechanically constructing them.

6 Bigraph Matching Algorithm

In this section, we show how to interpret the inference rules as functions—achieving an implementation proven correct in great detail.

In general, an inference tree can be divided into two parts: the twigs of the tree, consisting of all the rule applications above a SWX rule application, and the base of the tree, consisting of the remaining rule applications, cf. Figure 13. The base and the SWX rule applications determine the locations in the agent

![Figure 13: A sketch of the general structure of a normal inference tree for matching; dashed lines represent rules omitted for brevity](image)

at which the roots of the redex are matched, and each twig determines how the place graph subtree below a redex root is matched to the corresponding subtree in the agent. Each PAX leaf determines a component of the parameter, and each PAR leaf corresponds to a leaf node in the agent.

We turn the declarative matching specification of the preceding section into a matching algorithm by considering each rule operationally. By implementing the inference system faithfully rule by rule, we ensure that the proof of completeness is valid also for the implemented algorithm.

All rules of Figure 11 except SWX, CLO and DNF can be applied in two flavours: below and above a SWX rule. We distinguish these applications using primes (’) for rules applied above a SWX rule.

The DNF rule is concerned with normalising a and R, which is done using the algorithm described in Section 2.10, so for the remaining rules of the base of the tree we have as input to the algorithm wirings $\omega^a$, $\omega^R$ and discrete bigraphs $\bar{\beta}$ (agent) and $\bar{P}$ (redex). The goal is to find context wiring $\omega^C$, context term $\bar{Q}\pi$ and parameter term $\bar{q}$.

In general, there are zero or more matches, so in the implementation the application of each rule returns a lazy list of matches, each containing a context wiring, along with context and parameter bigraphs.

As the matching inference starts at the root of the inference tree, we will do the same, considering each rule of Figure 11 in turn, and giving an operational interpretation of it:

6.1 Rule Applications in the Base of the Tree

CLO: Operationally, CLO opens $\omega^a$ and $\omega^R$, producing $\sigma^a$ and $\sigma^R$ and assigning fresh names $Y_R$ to edges in $\omega^R$. 

15
In the implementation, we represent $\sigma$ as $\sigma^R \otimes \sigma^a$, where $\sigma^R$ contains all the outer names of $\omega^R$, and $\sigma^a$ contains the edges of $\omega^a$ for some $X_e$.

As we do not yet know which edges in $\omega^a : \rightarrow W$ match which names in $Y_R$, we represent $\sigma^a$ as $\alpha \sigma^R \otimes \sigma^a$, where $\sigma^a$ contains all the open links of $\omega^a$, and $\alpha$ is to be constructed during (i.e., returned by) the rest of the inference.

Inference must be done under the following condition:

- links in $\sigma^R$ must only be matched with links in $\alpha \sigma^a$

because redex edges can only match agent edges, not open links.

When the premise has been inferred, yielding $\text{id}_{Y_R} \otimes \sigma^C$, we determine $\sigma^C : \rightarrow W \uplus Y_C$ (as $Y_R$ is known), and then $Y_C = (W \uplus Y_C) \setminus W$. Finally, $\omega^C = (\text{id} \otimes /Y_C) \sigma^C$.

ABS: During the inference, ABS adds the links $\sigma^R_i$ to $\sigma^a_i$; the inner names of $\sigma^a_i$ are collected in a set $L$.

As $\sigma^a_i$ contains links bound in the agent, inference must be done under the following condition:

- links in $\sigma^a_i$ whose inner names are in $L$ must not be matched via $\sigma^R$, but must be matched via $\sigma$ (the local outer names of the redex) in the SWX rule.

This enforces the scoping rule for the resulting context bigraph.

When the premise of ABS has been inferred, $\sigma^C$ is computed by restricting the outer face of the context wiring to $W$, the outer names of $\sigma^a_i$.

MER: Taking $r$ be the outer width of redex $P_i$, we let $m = r + 1$, and compute all partitions of width $m$. Setting $m > r$ allows parts of the agent to not be matched by the redex ($l_i = 0$ in $\text{PAR}_{\sigma^R}^e$), that is, to become part of the context. For each partition, the premise is inferred, and if the permutation $\pi$ in the returned context really is a pushthrough of some permutation $\tau$, the factors $S_i$ of the tensor product are permuted before they are returned.

PER: For each $n$-permutation $\pi$, $\tilde{\pi}$ is computed by pushing $\pi$ through $\otimes_{\{i\}} Q_i$. After the premise has been inferred on the permuted redex primes, $\pi$ is used to permute the resulting parameter primes accordingly before they are returned.

PAR$_\sigma^C$: For each split of $m$ into $n$ parts, $\otimes_{\{i\}} P_j$ is computed, and after the premise has been inferred, the factors of the resulting parameter tensor product are concatenated into one tensor product before returning context and parameter.

PAR$_e^R$: For each $i$, the global outer names of $e_i$ is used to compute $\sigma^R_e \otimes \sigma^a_i$ by restriction, and similarly for $P_i$ and $\sigma^R_e \otimes \sigma^a_i$. The context wirings resulting from inferring the premise are combined using parallel product, but allowing inner name clashes as long as each operand maps the inner names to the same outer name.

ION: Letting $Y = \{i\}$, we split $\sigma^R_e = \sigma^R_e \parallel \sigma^a_i$ according to whether the inner names are in $Y$ or not; and similarly for $\sigma^a_i = \sigma^a_i \parallel \sigma^a_i$. Fresh names $\tilde{\sigma}$ are created.

### 6.2 Application of the SWX Rule

As the SWX rule preserves $\sigma^a$, its representation is not changed above the SWX rule. As CLO requires the resulting $\sigma^C$ to be of the form $\text{id}_{Y_R} \otimes \sigma^C$, where $\sigma^R : \rightarrow Y_R$, the third wiring above SWX is $(\text{id}_{Y_R} \otimes \sigma^C)((\text{id}_{Y_R} \otimes \sigma^R \otimes \sigma^R_{\omega^R}) \otimes \sigma^R \otimes \sigma^C) = \sigma^R \otimes \sigma^C (\text{id}_{Y_R} \otimes \sigma^R \otimes \sigma^R_{\omega^R})$. Letting $\sigma^C = \sigma \otimes \sigma^a$, we thus represent the context wiring above the rule by $\sigma^R \otimes \sigma^C (\text{id}_{Y_R} \otimes \sigma^a)$. 

```plaintext
[Aside: We cannot generally conclude there are no matches if the number of edges in $\omega^a$ is smaller than the number of edges in $\omega^R$! But the number of port-connected edges in $\omega^R$ must be greater than or equal to the number of edges in $\omega^a$. Unfortunately, port-connectedness is expensive to determine in the term representation.]
```
In the twigs we are thus given wirings $\sigma_e^a, \sigma_e^n, \sigma_R^a, \sigma_R^n, \sigma_C^a, \sigma_C^n$ and terms $\bar{p}$ (agent) and $\bar{P}$ (redex). The goal is to check that $\bar{p}$ matches $\bar{P}$, and find $\sigma^C, Z, \alpha$ and $\bar{q}$, as we want $\omega^a = \alpha \sigma_e^a \sigma_R^a, \omega^R = \sigma_R^a \sigma_R^R, \omega^C = \sigma_C^a \sigma_C^C$, and $\omega^R = \sigma_C^a \sigma_C^C (\text{id}_Z \otimes \sigma_n^C)$ in the judgment $\omega^a, \omega^R, \omega^C \vdash \bar{p}, \bar{q}$ above the SWX rule.

### 6.3 Rule Applications in the Twigs

**ABS’**: During the inference, the ABS’ rule adds links to $\sigma^C$. By adding them to $\sigma^C$, not $\sigma_n^C$, they are treated like internal edges in $\mathcal{R}$, and thus not linked to the parameter via $\text{id}_Z$.

**MER’**: For each $m$-permutation $\pi, \pi$ is computed; for each partition $\rho \in \bar{p}(n,m)$ of $n$ into $m$ (possibly empty) subsets, the tensor product of $m_j$’s are computed, and then the premise is inferred, returning $\sigma^C, \alpha$ and $\bar{q}$.

**PER’**: The premise is inferred, and the resulting $\bar{q}$ are permuted using $\bar{\pi}$ before they are returned.

**PAR’**: The premise is inferred, and the resulting list of $n$ tensor products are concatenated and returned as one product $\otimes^m_i q_i$.

**PAR’**: Taking PAR’ literally, $\sigma^a$ and $\sigma^C$ must be split when performing the subinferences. However, as the inner face of $\sigma^a$ must always match the global outer face of $\alpha$, explicit splitting of $\sigma_e^a, \sigma_e^n, \sigma_R^a, \sigma_R^n, \sigma_C^a, \sigma_C^n$ can be avoided. This also implies that (1) need only be solved for links mapping outer names of $\sigma$ (i.e., $X \oplus Z$).

**ION’**: Deconstructing agent and context ions, their controls are checked for equality; for each $u_i \in \text{dom}(\sigma^C_e)$ we update $\alpha'$ so that $\alpha' \sigma^a_e(y_i) = \sigma^C_e(u_i).$ Using a fresh $\bar{v}$, the premise is inferred, and $\alpha'$ and $\sigma^C$ are updated for each $u_i \notin \text{dom}(\sigma^C_e)$ so that $\sigma^C\alpha'(\sigma^C_e(u_i))$ is equal to $\alpha' \sigma^a_e(y_i)$ or $\sigma^a_n(y_i)$, depending on whether $y_i \in \text{dom}(\sigma^C_e)$ or not.

**PAX’**: At the PAX’ rule, we are given $V, (X \oplus Z)$ and $\alpha$, and $\sigma^a$ as $\alpha' \sigma^a_e \otimes \sigma^a_n$, and $\sigma^C$ as $\sigma^C_e \otimes \sigma^C(id_Z \otimes \sigma^C_n)$. We must now solve the equation $\sigma^a = \sigma^C(id \otimes \alpha \tau)$, i.e.

$$\alpha' \sigma^a_e \otimes \sigma^a_n = (\sigma^C_e \otimes \sigma^C(id_Z \otimes \sigma^C_n))(id_Z \otimes \alpha \tau)$$

for $\alpha', \sigma^C, Z$ and $\tau$, where $X_e \cap Z = \emptyset$ (recall $\sigma^C_e : X_e \rightarrow$).

### 7 Nondeterminism

Given these term-based rules and the normal inference grammar, proven correct matching has been expressed in an operational, that is, implementable, form. However, there is still a fair amount of nondeterminism left, but fortunately we can clearly identify where it occurs:

**Grammar selection**: Which branches to select for $\mathcal{P}_G, \mathcal{P}_P, \mathcal{G}_G$ and $\mathcal{P}_P$.

**Tensor grouping**: How to group the tensor product in $\mathcal{P}$.

**Children partitioning**: How to partition molecules in $\mathcal{E}$.

**Prime permutation**: How to permute redex primes in $\mathcal{P}$.

**Context-redex-parameter wiring**: How to choose $Z, \alpha$ and $\tau$ in $\mathcal{P}$.

**Mapping closed links**: How to find an appropriate decomposition of agent wiring in $\mathcal{L}$ such that closed agent links are matched correctly with closed redex links (i.e., determining $\sigma^a$ and $\mathcal{Y}_a$).

When implementing matching, the challenge is to develop a heuristic that will handle typical cases well. In general, an agent-redex pair can lead to many different matches, so in our implementation we return for every inference rule a lazy list of possible matches.
To handle nondeterminism, we return possible matches as follows, bearing in mind that operationally speaking, rules applied below SWX are given agent and redex, while rules above SWX are given agent (, redex) and context:

Grammar selection: For $D_G$ and $D_P$, we concatenate the returned lazy lists returned from matching each branch in turn. However, if $PAX$ succeeds, there is no reason to attempt a SWX match, as no new matches will result.

For $D_G'$ and $D_P'$, we try each branch in turn, returning the first branch that succeeds, as later branches will not find any new matches.

Tensor grouping: For given $m$ and $n$ in $\text{PAR} \equiv$, we compute all the ways of splitting $[0, \ldots, m-1]$ into $n$ (possibly empty) subsequences, trying out matching for each split. Note that this need only be done for applications of $\text{PAR} \equiv$ below the SWX rule.

Children partitioning: For given $m$ and $n$ in $\text{MER}$, we compute all the ways of partitioning $\{0, \ldots, m-1\}$ into $n$ (possibly empty) sets, trying out matching for each partitioning.

Prime permutation: For given $n$ in $\text{PER} \equiv$, we compute all $n$-permutations, trying out matching for each permutation. This is done for applications of $\text{PER} \equiv$ below the SWX rule; above, similar permutations are computed in the MER rule.

Context-redex-parameter wiring: Given global agent wiring, we compute the ways of decomposing it into $\sigma(id_2 \otimes \alpha \tau)$, returning a match for each decomposition.

Mapping closed links: We split agent wiring into named and closed links, and postpone the actual mapping of each closed link to redex or context links until some constraint, given by $\text{ION}$ or $\text{PAX}$ produces it.

Note that even after limiting nondeterminism in this way, we can still in general find several instances of the same match, reached by different inference trees, as we are computing abstract bigraph matches using concrete representations. For instance, matching redex $R = K_1$ in agent $a = \text{merge}(K_1 \otimes K_1)$ produces matches with context $C_1 = \text{merge}(id_1 \otimes K_1)$ and context $C_2 = \text{merge}(K_1 \otimes id_1)$.

8 Tool Implementation and Example Modelling

We have implemented a BPL Tool as a reference implementation of binding bigraph matching, and as a toolbox for experimenting with bigraphs. It is written in SML, consists of parser, normalisation and matching kernel, and includes web and command line user interfaces [4].

To ensure correctness, we have implemented normalisation, renaming, regularisation and matching faithfully by implementing one SML function for every inference rule—in the case of matching, two: one for applications above and one for below the SWX rule.

The BPL Tool handles normalisation, regularisation, matching and reaction for the full set of binding bigraphs, and allows construction of simple tactics for prescribing the order in which reaction rules should be applied. The following example output is taken verbatim from the command line interface, which is based on the SML/NJ interactive system; omitted details are indicated by “[...]”.

As an example, we model the polyadic $\pi$ calculus, running the mobile phone system introduced in Milner’s $\pi$ book [17]. The calculus can be modeled by a family of reaction rules $\{\text{REACT}_i | i = 0, 1, \ldots\}$, one for each number of names that are to be communicated in a reaction [13]; $\text{REACT}_2$ is shown in Figure 14.

The signature for the nodes modelling the calculus and the mobile phone system is constructed using passive and atomic functions as shown in Figure 15. For this system, we only need $\text{Send}$ and $\text{Get}$
Figure 14: \(\pi\) calculus reaction rule shown as bigraphs and BPL value.

\[
\text{REACT}_2: (\bar{x}(y_1, y_2).P_0 + P_1) | (x(z_1, z_2).P_2 + P_3) \rightarrow \{z_i/y_i\}P_0 | P_2
\]

\[
\text{val REACT2} = \"REACT2\" :::: \text{Sum o (Send2}[x,y_1,y_2]\ ']' idp(1)) ']' Sum o (Get2}[x][z_1],[z_2]\ ']' idp(1))
--[0 \rightarrow 0, 1 \rightarrow 2]--|>
(y_1/z_1 * y_2/z_2 * x//[] * idp(1)) o (idp(1) ']' '[z_1, z_2]'\);

Figure 15: Signature for \(\pi\) calculus and mobile phone system nodes.
nodes for $\text{REACT}_0$ and $\text{REACT}_2$. Note that all reaction rule nodes are passive, preventing reaction within a guarded expression.

The system consists of a car, one active and one idle transmitter, and a control centre, as shown in Figure 16. Internally, a prime product constructed using the ‘$|$’ operator is represented by a wiring and

```plaintext
- val System1 = simplify (  
  Car[talk1,switch1]  
  $'|'$ Trans[talk1,switch1,gain1,lose1]  
  $'|'$ Idtrans[gain2,lose2]  
  $'|'$ Control[lose1,talk2,switch2,gain2,lose2,talk1,switch1,gain1]);
```

Figure 16: Definition of the mobile phone system, $\text{System}_1$

merge$_2$ composed with a binary tensor product. The function simplify applies various heuristics for producing human-readable bigraph terms, in this case for a prime product of four factors.

The definition of these nodes and connections, shown in Figure 17, allows the control centre to switch $\text{Car}$ communication between the two transmitters (supposedly when the car gets closer to the ilde than the active transmitter), and allows the car to talk with the active transmitter. Note that in the BPL tool, we define a node by a rule that unfolds an atomic node into a bigraph corresponding to the defining $\pi$ calculus expression.

Our BPL definition of the initial system in Figure 16, $\text{System}_1$, is the folded version; as BPL matching is complete, querying the tool reveals the four possible unfolding matches, illustrated in Figure 18. Here mkrules constructs the internal representation of a rule set, and print_mv prettyprints a lazy list of matches, produced by the matches function.

Using react_rule that simply applies a named reaction rule, and ++ that runs its arguments sequentially, we construct a tactic, TAC_unfold, for unfolding all four nodes once, shown in Figure 19. Applying this tactic using function run, we get an unfolded version of the system.

Using the BPL Tool for all possible matches in the unfolded system reveals exactly the switch and talk actions, initiated by $\text{REACT}_2$ and $\text{REACT}_0$ rules, respectively, cf. Figure 20. Applying the $\pi$ calculus reaction rules for switching, we arrive at $\text{System}_2$, where $\text{Car}$ communication has been switched to the other transmitter, as witnessed by the outer names to which $\text{Car}$ ports link, as well as the order of names to which $\text{Control}$ ports link.

This concludes our description of the example highlighting how we can use the BPL Tool to experiment with bigraphical reactive systems.
Figure 17: Definitions of Car, Trans, Idtrans and Control nodes.

Figure 18: Determining which rules match System1.
- val TAC_unfold = 
  react_rule "DEF_Car" ++ react_rule "DEF_Trans" ++
  react_rule "DEF_Idtrans" ++ react_rule "DEF_Control";

- val System1_unfolded = run rules TAC_unfold System1;

val System1_unfolded =

(lose1//[lose1_3f9, lose1_419, lose_441, lose_459, lose_45d]
  * talk2//[talk2_3f8, talk2_40f, talk2_418]
  * switch2//[switch2_3f7, switch2_40e, switch2_417]
  * gain2//[gain2_3f6, gain2_410, gain_431, gain_438]
  * lose2//[lose2_3fd, lose_430]
  * talk1//[talk1_3fc, talk_460, talk_465, talk_482, talk_485]
  * switch1//[switch1_3fb, switch_447, switch_45f, switch_480, switch_481]
  * gain1//[gain1_3fa, gain_442, gain_45e]) o merge(4) o

(Sum o merge(2) o
  (Send0[talk_485] o Car[talk_482, switch_481] *
    Get2[switch_480]([[t_47d], s_47c]) o
    (<<s_47c, t_47d> Car[t_47d, s_47c]))) *

Sum o merge(2) o
  (Get0[talk_465] o Trans[talk_460, switch_45f, gain_45e, lose_45d] *
    Get2[lose_459]([[t_446], s_445]) o
    (<<s_445, t_446>
      Sum o (Send2[switch_447, t_446, s_445] o Idtrans[gain_442, lose_441])))) *

Sum o Get2[gain_438]([[t_433], s_432]) o
  (<<s_432, t_433> Trans[t_433, s_432, gain_431, lose_430]) *

Sum o
  (Send2[lose1_419, talk2_418, switch2_417] o
    (Sum o
      (Send2[gain2_410, talk2_40f, switch2_40e] o
        Control[lose2_3fd, talk1_3fc, switch1_3fb, gain1_3fa, lose1_3f9,
          talk2_3f8, switch2_3f7, gain2_3f6])))

: 0 -> <<lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1>> : agent

Figure 19: Unfolding System1, using the TAC_unfold tactic.
9 Conclusion and Future Work

We have developed a provably sound and complete inference system over bigraph terms for inferring legal matches of bigraphical reactive systems. Moreover, we have implemented our BPL Tool, the first implementation of bigraphical reactive systems. We have demonstrated a simple, but concrete, example of how the tool can be used to simulate bigraphical models. We have found it very useful to base this first implementation of bigraphical reactive systems so closely on the developed theory—this has naturally given us greater confidence in the implementation, but the implementation work has also helped to debug the developed theory.

There are lots of interesting avenues for future work. While the current implementation of BPL Tool is efficient enough to experiment with small examples, we will try to make it more efficient by using a number of different techniques: we plan to investigate how to prune off invalid matches quickly, for instance by making use of sorting information [3]. Moreover, we will investigate to what extent we can capture the link graph matching via a constraint-based algorithm.

We also plan to investigate smarter ways of combining matching and rewriting. As a starting point, we have made it possible for users to combine tactics to inform the tool in which order it should attempt to apply reaction rules.

Jean Krivine and Robin Milner are currently investigating stochastic bigraphs, which will be particularly important for simulation of real systems. We hope that our detailed analysis of matching for binding bigraphs will make it reasonably straightforward to extend it to stochastic bigraphs.

Acknowledgements

The authors would like to thank Mikkel Bundgaard and anonymous referees for detailed comments to earlier versions of the paper.


References


A Auxiliary Technologies Details

A.1 Normalising

We define a normalisation relation $t \downarrow_B t'$ for elementary bigraphs: $merge_n[\Gamma X], y/X, K_{\gamma} \vec{X}$ and $\pi$ as shown in Figure 21, and inductively for operations: abstraction $(X)P$, product $\otimes_i B_i$ and composition $B_1B_2$ as shown in Figure 22, where the notation $\sigma|^Y \{X \rightarrow y \in \sigma \ | \ y \in Y\}$.

\[
\begin{array}{c}
B_{\text{mer}} & N \equiv \langle 0 \rangle (id_0 \otimes merge_n) \bigotimes_{i \in n} \{ \Gamma \mid id_0 \} & P \equiv (id_0 \otimes \langle \_ / \_ \rangle)_i N & D \equiv id_0 \otimes (\bigotimes_{i \in 1} P)id_n \\
 & \quad \downarrow_B (id_0 \otimes id_{[0]})D \\
B_{\text{con}} & N \equiv \langle 0 \rangle (id_X \otimes merge_1) \bigotimes_{i \in 1} (id_X \otimes id_1)^\Gamma \Gamma Y & P \equiv (id_X \otimes \langle \_ / \_ \rangle)_i N & D \equiv id_0 \otimes (\bigotimes_{i \in 1} P)id_{[X]} \\
 & \quad \downarrow_B \{ \Gamma \mid id_X \otimes id_{[0]} \}D \\
B_{\text{bir}} & Y = \{ \vec{y} \} & M \equiv (id_0 \otimes K_{\gamma} \vec{Y}, X) (id_X \otimes merge_1) \bigotimes_{i \in 1} (id_X \otimes id_1)^\Gamma \Gamma X & N \equiv \langle 0 \rangle (id_Y \otimes merge_1) \bigotimes_{i \in 1} M & P \equiv (id_Y \otimes \langle \_ / \_ \rangle)_i N & D \equiv id_0 \otimes (\bigotimes_{i \in 1} P)id_{[X]} \\
 & \quad \downarrow_B \{ \Gamma \mid id_Y \otimes id_{[0]} \}D \\
B_{\text{per}} & Y_i = \{ \vec{y_i} \} & N_i \equiv \langle Y_i \rangle (id_{\vec{y_i}} \otimes merge_1) \bigotimes_{i \in n} (id_{\vec{y_i}} \otimes id_1)^\Gamma Y_i & P_i \equiv (id_0 \otimes \vec{y_i}/\vec{y_i})N_i & D \equiv id_0 \otimes (\bigotimes_{i \in m} P_i)\pi \\
 & \quad \downarrow_B \{ m, \vec{X}, X \} \rightarrow \{ m, \vec{Y}, X \} \downarrow_B (id_0 \otimes id_{\vec{y}})D \\
\end{array}
\]

Figure 21: Inference rules for normalising elementary bigraph expressions.
\[ \begin{align*}
\text{Babs} & \quad W^X = \{ \vec{W}^X \}, \quad W^\vec{X} = \{ \vec{W}^\vec{X} \}, \quad U = \{ \vec{z}^X \}, \quad N = (W^X \cup \vec{W}^\vec{X})G \quad P \equiv (\text{id}_{W^X} \otimes \vec{z}^X \otimes \vec{W}^\vec{X})N \\
\quad \quad (X)b \downarrow_{\text{B}} (\vec{z}^X / W^X \otimes \text{id}_{(\vec{1}_j)}) (\text{id}_{\bar{b}} \otimes (\otimes_{i \in n} \text{id}_{z_i} \otimes \vec{X}) (WG) \text{id}_{\bar{b}}) \\
\quad b_1 \downarrow_{\text{B}} (\alpha_1 \otimes \text{id}_{(\vec{1}_j)}) D_1 : \langle m', \vec{X}', X' \uplus Z' \rangle \rightarrow \langle n, \vec{U}', U' \uplus W \rangle \\
\quad b_2 \downarrow_{\text{B}} (\alpha_2 \otimes \text{id}_{(\vec{1}_j)}) D_2 : \langle m, \vec{X}, X \uplus U \rangle \rightarrow \langle m', \vec{U}', U' \uplus Z \rangle \\
\quad D_1 \equiv \alpha_1 \otimes (\otimes_{i \in n} P^1_1) \pi_1 : \langle m', \vec{X}', X' \uplus Z' \rangle \rightarrow \langle n, \vec{U}', U' \uplus V' \uplus W' \rangle \\
\quad D_2 \equiv \alpha_2 \otimes (\otimes_{i \in m} P^2_2) \pi_2 : \langle m, \vec{X}, X \uplus U \rangle \rightarrow \langle m', \vec{U}', U' \uplus V \uplus W' \rangle \\
\quad P^1_1 : \langle m', \vec{X}', X' \rangle \rightarrow \langle (U'_1), U'_1 \uplus V'_1 \rangle \\
\quad P^2_2 : \langle m, \vec{X}, X \rangle \rightarrow \langle (U'_2), U_2 \uplus V'_2 \rangle \\
\quad \omega_1 : V'_1 \uplus W' \rightarrow W \\
\quad \omega_2 : V_2 \uplus W \rightarrow Z \\
\quad \alpha_1 : Z \rightarrow W \quad \alpha_2 : U \rightarrow W \\
\quad V_2 = \biguplus_{i \in m} V''_2 \\
\quad \otimes_{i \in m} P^2_2 \pi_1 = \otimes_{i \in n} \vec{P}_i. \\
\quad \pi = \underline{\pi}_1 \underline{\pi}_2 \\
\quad \sigma_1 = \text{id}_{U} \otimes (\otimes_{i \in n} \sigma_i) \quad \omega = \omega_0 \otimes \alpha_2 \otimes \text{id}_{\vec{V}'_1} \otimes \text{id}_{V'} \\
\quad \pi = \pi_1 \pi_2 \\
\quad \sigma = \text{id}_{U} \otimes (\otimes_{i \in n} \sigma_i) \\
\quad D \equiv \text{id}_{U} \otimes (\otimes_{i \in n} \pi_i) \\
\quad b_1 b_2 \downarrow_{\text{B}} (\omega \otimes \text{id}_{(\vec{1}_j)}) D
\end{align*} \]

Figure 22: Inference rules for normalising bigraph abstraction, product and composition expressions.
A.2 Renaming

Let a link namer be a map $\mu$ mapping every link $l$ (outer name or edge) in its domain to a pair $(E, X)$, where $E$ is a set of names used internally to compose the link, and $X$ are the inner names linking to $l$. We let $\mathcal{Y}(Y, \mu) = \bigcup_{\nu, \rho \rightarrow (X_0, X_1) \in \mu} X_1$ and define link namer composition by

$$\mu_1 \circ \mu_2 = \{ y_1 \mapsto (E_1 \cup X_1 \cup V_1, V_2) \mid y_1 \mapsto (E_1, X_1) \in \mu_1 \land V_i = \mathcal{Y}(X_1, \mu_2) \}$$

essentially composing links of $\mu_1$ with those of $\mu_2$, and adding closed links from $\mu_2$.

We then define a function $\text{linknames}$, mapping terms to link namers, by the equations given in Figure 23. By using the link namers of immediate subterms, we can determine whether a term can be normalised without name clashes. To this end, we define a predicate $\text{normalisable}$ by the equations given in Figure 24. We basically just require, that at no level in the term does two different links share any internal names.

$$\text{normalisable} = \text{true}$$

Figure 23: Function for determining which names are used internally to compose a link

Figure 24: Function for determining whether a (well-formed) term is normalisable

Renaming is achieved by the judgment $U \vdash \alpha, t \downarrow t', \beta \vdash V$, where $U$ is a set of used names and $\alpha$ renames $t$’s inner names to those of $t'$, while $\beta$ renames $t$’s outer names to those of $t'$ and $V$ extends $U$ with names used in $t'$. The system of rules for inferring this judgment is given in Figure 25.
### A.3 Regularising

The system of rules for inferring a permutation-free term representing a regular bigraph is given in Figure 26.

**Figure 26:** Removing nontrivial permutations from regular bigraphs.

\[
\begin{align*}
\text{Rmer} & : U \vdash \text{id}_\emptyset, \text{merge}_\emptyset \Downarrow \beta \text{ merge}_\emptyset, \text{id}_\emptyset \Downarrow U \\
\text{Rwir} & : Z = \{z\} \quad Z \cap U = \emptyset \quad |Z| = |z| = |y| \\
\beta = \{y_i \mapsto z_i\} \quad \overline{X}' = \alpha(\overline{X}) \\
\text{Rper} & : U \vdash \alpha, \gamma \Downarrow \beta, \gamma \Downarrow U \cup Z \\
\text{Rcom} & : \alpha' = \alpha(X) \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Rcom</th>
<th>$X' = \alpha(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$U \vdash \alpha' \uparrow \beta \Downarrow X', \alpha \Downarrow U$</td>
</tr>
</tbody>
</table>

| Rtm   | $\alpha_i = \alpha |_{X_i}$ |
|-------|------------------|
|       | $U_i \vdash \alpha_i, t_i \Downarrow \nu |_{X_i}, \beta_i \Downarrow U_{i+1}$ |

<table>
<thead>
<tr>
<th>Rcom</th>
<th>$\beta = \bigotimes_{i \in \emptyset} \beta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$U_0 \vdash \alpha, \bigotimes_{i \in \emptyset} t_i \Downarrow \nu, \beta \Downarrow U_n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rtm</th>
<th>$t_i : (m_i, \overline{X}_i, X_i) \rightarrow J_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$U_1 \vdash \alpha_1, t_1 \Downarrow \nu</td>
</tr>
</tbody>
</table>

| Rcom  | $\nu \Downarrow \beta_1 \Downarrow \nu |_{X_1}$ |
|-------|------------------|
|       | $U_2 \vdash \beta_1, t_1 \Downarrow \nu |_{X_1}, \beta_2 \Downarrow V_2$ |

\[
\begin{align*}
\text{A.3} & : \text{Regularising} \\
\end{align*}
\]