Electronic Communications of the EASST
Volume 61 (2013)

Selected Revised Papers from the 4th International Workshop on Graph Computation Models (GCM 2012)

Big Red: A Development Environment for Bigraphs

Alexander Faithfull, Gian Perrone, Thomas T. Hildebrandt

10 pages

Guest Editors: Rachid Echahed, Annegret Habel, Mohamed Mosbah
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Big Red: A Development Environment for Bigraphs

Alexander Faithfull[1], **Gian Perrone**[2], **Thomas T. Hildebrandt**[3]

[1] alef@itu.dk
[2] gdpe@itu.dk
[3] hilde@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Abstract: We present Big Red, a visual editor for bigraphs and bigraphical reactive systems, based upon Eclipse. The editor integrates with several existing bigraph tools to permit simulation and reachability analysis of bigraphical models. We give a brief introduction to the bigraphs formalism, and show how these concepts manifest within the tool using a small motivating example developed in Big Red. We go on to outline its architecture and implementation, and comment on possible future work.

Keywords: bigraphs, editor, reactive systems

1 Introduction

Bigraphical reactive systems are a class of graph-rewriting systems designed to capture orthogonal notions of *connectivity* and *locality* through the use of two graph structures—a *place* graph, and a *link* graph. They were first proposed by Robin Milner [Mil09] to address the challenges associated with modelling of ubiquitous computing applications. Bigraphs have been successful in capturing the syntax and semantics of a number of well-known formalisms (e.g., π-calculus [Jen06], Petri Nets [LM06], CCS [Mil06], and many more), as well as more diverse applications such as business processes [HNO06], biological systems [DK08], wireless networks [CS12], and applications for context-aware systems [BDE+06].

The Big Red tool is a prototype editor to support the development of bigraphs and bigraphical reactive systems in a visual manner. It interfaces with existing bigraph tools such as the BigMC bigraphical model checker [PDH12] to permit the execution of models. Big Red aims to make bigraphs more accessible to novice users, as well as providing development support to more experienced bigraph users. Bigraphs have a visual presentation that is formal and unambiguous, and one of the major benefits is the ability to present a relatively complex bigraphical model in a way that is comprehensible by non-experts. This is the motivation for the development of Big Red: making it easier to create and interact with bigraphs increases the applicability and utility of the formalism in more diverse application areas.

Below we first briefly in Section 2 describe the previous efforts to implement bigraphical reactive systems. We then proceed in Section 3 describing how bigraphs are expressed in Big Red, using a small example of a context-aware printing system, inspired by that given in [BDE+06]. Finally, Section 4 briefly describes the implementation of the tool, and suggests ways in which it may be extended using additional modules.

2 Related Work

To date, bigraphical reactive systems have been largely implemented using term representations of bigraphs, such as that used in the BPLtool [GDBH07], or in the BigMC model checker [PDH12]. (While the BPLtool does support graphical visualisation of its *results*, its *input* can only be given textually.) While this is appropriate for bigraph experts, it presents a significant learning curve for novice users, and ignores the benefits provided by the formal graphical syntax provided by bigraphs.

One of the first attempts at creating a graphical editor for bigraphs was within the Bigraphspace [Gre09] project in 2009, during which a prototype bigraph editor—also based upon Eclipse—was developed; however, this work was never completed, and no usable editor currently exists. Bigraphspace used the correspondence between the structure of bigraphs and XML documents [HNO06] to provide a tuplespace-like API with which to manipulate bigraphs.

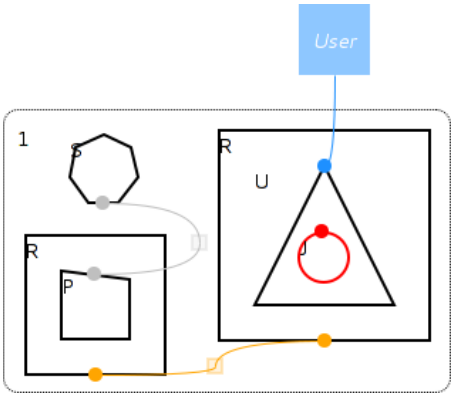
3 Bigraphs in Big Red by Example

Below we describe bigraphs and how to construct them within Big Red using an example of a bigraphical reactive system: a context-aware printing system, inspired by that given in [BDE+06]. This system describes a building in which users can submit print jobs to a print spool, and then move into a room with any printer connected to that print spool, at which point the printer will complete the job. Fig. 1 shows Big Red's graphical and internal representations of a bigraph of this system. It contains a single print spool (shown as a heptagon) and two rooms (squares) connected by a door (the yellow link); the left room contains a printer which is linked to the print spool, and the right room contains a user (a triangle) with the identity `User`. The user has a print job (a circle) which has not yet been submitted to the spool.

In more detail, a bigraph consists of two graphs: the *place graph* and the *link graph* (hence the name *bi-graph*). The place graph is a forest of labelled trees, the roots of which are indexed by integers (which Milner referred to as *regions*). The place graph parent relationship is indicated by nesting of nodes in the graphical syntax. The example bigraph in Fig. 1, with one region, uses the place graph to represent both physical nesting (rooms contain users and printers) and logical nesting (users contain their print jobs).

Each of the non-root nodes of the place graph is assigned a label, called its *control*, which is drawn from the bigraph's *signature*. Our example defines five controls: *room* (`R`), *printer* (`P`), *spool* (`S`), *job* (`J`) and *user* (`U`). We will use "`X` node" to mean a node that is labelled with the control `X`. The control of a node also defines the number of *ports* of the node, referred to as the *arity* of the control and provided by a function *ar* given as part of the signature. All controls in this example have arity 1, meaning that every node labelled with a control has a single port.

Each port is mapped to a *name*, which can be thought of as an attribute of a node. Names can either be *global* or *local* to a bigraph. A global name is represented by placing the name in the set of so-called *outer names* of the bigraph, and providing a link to that name from every port to which it is assigned. The example bigraph has a single global name, `User`, representing the (globally accessible) identity of the user. A local name is simply represented by providing a link to an *edge* from every port to which it is assigned. The example bigraph has two edges,




```
<bigraph xmlns="...">
  <signature xmlns="..." src="..." />
  <edge name="a"/>
  <edge name="b"/>
  <outername name="User"/>
  <root name="1">
    <node control="Room" name="b">
      <port link="b" name="door"/>
      <node control="Printer" name="e">
        <port link="a" name="spool"/>
      </node>
    </node>
    ...
```

Figure 1: An example bigraph expressed in Big Red's visual and XML syntaxes.

one connecting the two rooms, and one connecting the printer to the spool. The links between names, ports, and edges together constitute the link graph, which can be viewed as an undirected hypergraph.

A key feature of bigraphs is that they can be composed as *contexts*; this is facilitated in the place graph by allowing some of the leaves to be *holes* (also called *sites* by Milner), which—like the roots—are indexed by integers, and in the link graph by partioning the global names into outer names and *inner names*.

Intuitively, a bigraph A with n holes is a context in which a bigraph B with n roots (regions) can be inserted, the result of which is a bigraph $A \circ B$ in which the ith hole of A is replaced by the nodes nested inside the ith region of B, so that the holes of A and roots of B are deleted. The set of inner names of bigraph A must be equal to the set of outer names of B, as each outer name of B is re-linked in the composed bigraph $A \circ B$ to the global—or local—name of A linked to the corresponding inner name.

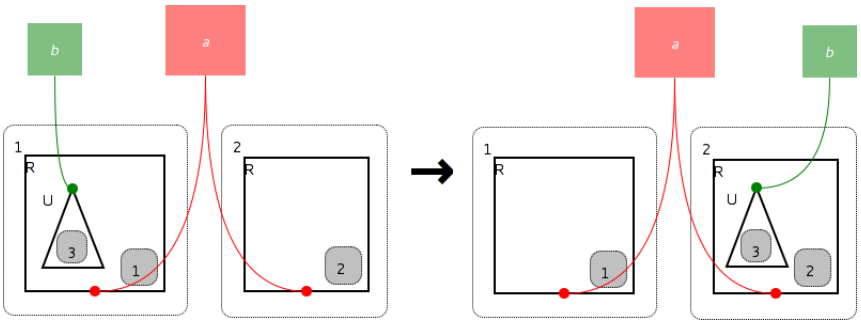
An important use of bigraphical composition is in defining the dynamic behaviour of a bigraphical reactive system as a set of *reaction rules*. (The reaction rules for our example system are shown in figures 2–5.) A reaction rule is a pair of bigraphs connected by an arrow from left to right, $A \to B$. The intuition of a reaction rule is that if the left hand bigraph A *matches* a sub bigraph inside the bigraph S representing the system, then a reaction can occur, replacing A by B. A bigraph A matches a sub bigraph inside S if S can be decomposed as $S = C \circ A \circ D$, i.e. A placed in a context C, and a bigraph D (the parameters of the rule) placed in the holes of A, represented as indexed, shaded place graph leaf nodes. If A and B have the same hole indexes the bigraph resulting from the reaction is $C \circ B \circ D$. The reaction can only occur if none of the holes in C are nested within nodes assigned a *non-active* control. Whether a control is active or non-active is defined in the signature and is used to restrict where reactions can occur in a system. Only the room control must be active in our example; we explain why below.

The constituents of a bigraph are defined formally by Milner [Mil09] as a 5-tuple:

$$(V, E, ctrl, prnt, link) : \langle n, X \rangle \to \langle m, Y \rangle$$

where V is the set of nodes, E is a set of edges, $ctrl : V \to \Sigma$ assigns controls to nodes drawn from a signature Σ, $prnt : n \uplus V \to V \uplus m$ is the parent map defining the place graph nesting (where n is the indexes of holes and m is the index set of regions), and $link : X \uplus P \to E \uplus Y$ is the link map, where X is the set of inner names, $P = \{(v,i) : v \in V \wedge i \in 0 \ldots ar(ctrl(v))\}$ is the set of all ports, and Y is the set of outer names. Together, we refer to $\langle n, X \rangle$ as the inner interface of the bigraph and $\langle m, Y \rangle$ as the outer interface.

We explain the four reaction rules that define the behaviour of the context-aware printing system below. Big Red allows for the definition of reaction rules, but does not itself perform reactions; however, it can import from and export to the term language of the BigMC tool, which *can* execute reaction rules. Below the graphical representations of the reaction rules, we additionally give their representation in this term language.

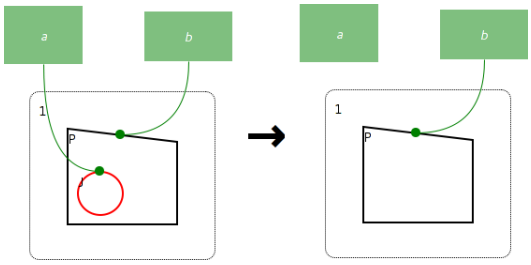



```
Room[a].(User[b].$3 | $1) || Room[a].$2 -> Room[a].$1 || Room[a].($2 |
User[b].$3)
```

Figure 2: The MoveRoom rule.

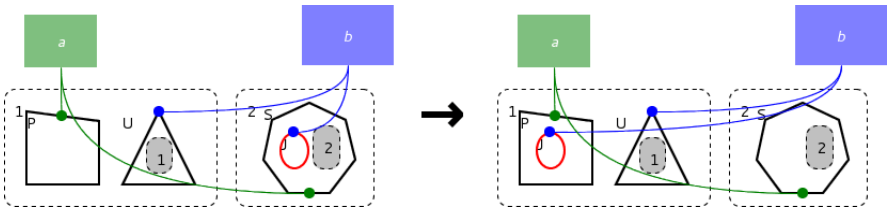
Fig. 2 shows the MoveRoom rule, which matches a user (represented by a `U` node), and selects any two rooms that are connected by some name `a` (which may be any name in common, as the context may freely rename or alias outer names). It then (in the right-hand bigraph) specifies that the user will move to the other room, leaving the contents of each room otherwise unchanged—the contents of the rooms, captured by the holes 1 and 2, and anything the user is carrying, captured by hole 3, are preserved between the left and the right hand side. We can find a match for this rule in Fig. 1; the context contains the spool and two holes, and the parameters consist of the empty bigraph (captured by hole 1), the printer (captured by hole 2) and the user's print job (captured by hole 3).

Fig. 3 shows the FinishJob rule, which captures the idea of a printer finishing a job (by executing the actual print process), and the job is therefore removed completely from the system by this rule, disconnecting it from its associated user. (In our example system, printers are always contained in rooms, and so the context in any match of this reaction rule necessarily includes the room that contains the printer. As a consequence, the room control must be active to allow the reaction to take place.)




```
Printer[b].Job[a] -> Printer[b]
```

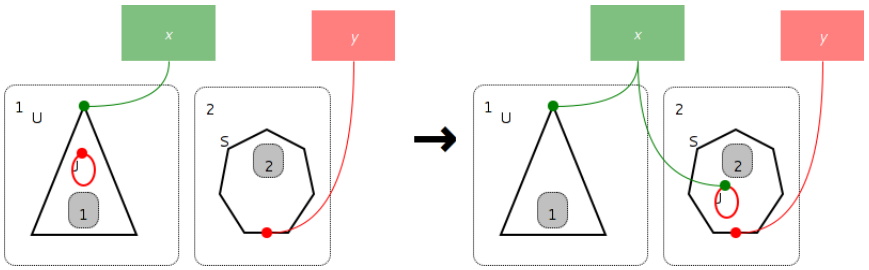
Figure 3: The FinishJob rule.




```
Printer[a] | User[b].$1 || Spool[a].($2 | Job[b]) -> Printer[a].Job[b]
| User[b].$1 || Spool[a].$2)
```

Figure 4: The JobToPrinter rule.

Fig. 4 shows the JobToPrinter rule, which transfers a job from the spool to a printer (represented by a `P` node) that is co-located with the user associated with that job. (Notice that printers may only contain one job at a time.)




```
User[x].($1 | Job[-]) || Spool[y].$2 -> User[x].$1 || Spool[y].($2 |
Job[x])
```

Figure 5: The JobToSpool rule.

Fig. 5 shows the JobToSpool rule, which allows a print job (represented by a `J` node) to be transferred from a user (represented by a `U` node) to a spool (represented by an `S` node), adding an identifying link to connect users to their submitted print jobs. We can also find a match for this rule in Fig. 1; the context contains a hole, the room containing the printer, and the other room containing a hole, and both the parameters consist of the empty bigraph (as neither the user nor

the spool have any other print jobs).

Interested readers are referred to [Mil09] for a more detailed description of bigraphs and bigraphical reactive systems.

4 Implementation

4.1 Architecture

Big Red is implemented as a number of Eclipse *plugins*, which extend the Eclipse platform with additional file formats representing the objects of a bigraphical reactive system, wizards to create model files, and editors to modify them. In turn, Big Red defines several Eclipse *extension points*: these allow other plugins to contribute extensions to Big Red, adding support for new external tools, export formats, and variants of the bigraphical model.

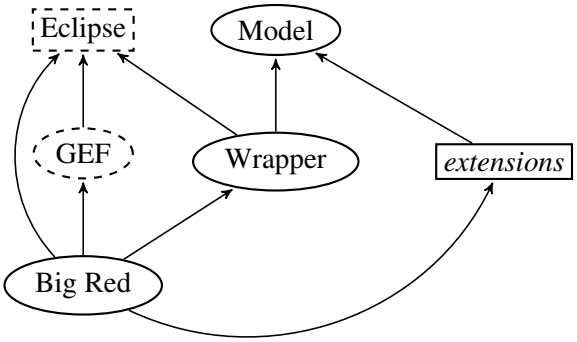


Figure 6: An architectural overview of Big Red.

As an overview of Big Red's architecture, Fig. 6 gives its dependency graph. Circles represent plugins, and rectangles represent groups of plugins: those with dashed outlines are part of the Eclipse platform, while those with solid outlines make up Big Red.

Most of Big Red's dependencies are simply parts of the basic Eclipse platform—the user interface and the workspace and resource management code. Support for building modelling tools is added by the *Graphical Editor Framework* (GEF), a specialised toolkit for implementing model-view-controller-based editors; Big Red's bigraph and reaction rule editors are both built on top of this component. (The other contributions to the UI are made using *JFace* and *SWT*, both standard Eclipse components.)

Quite apart from its specific support for modelling tools, Eclipse is a very portable and widely-used platform with an active community and many pre-built features, which makes it an ideal choice for the rapid development of an editor.

Independent but integrated. The implementation of the bigraphical model is actually split between *two* plugins. The bigraphical model itself (and all of its supporting infrastructure) is not a true Eclipse plugin—it has no dependencies upon Eclipse, contributes no extensions, and

exposes no extension points, and so it can also be used outside of the Eclipse platform. This plugin also implements the extensibility mechanisms, but it does not expose them as extension points—this is the job of the second plugin, which wraps those mechanisms in Eclipse concepts.

4.2 The user interface

Big Red uses two different Eclipse components to construct its contributions to the user interface: SWT and GEF. The bigraph editor, which uses GEF, and the signature editor, which uses SWT and JFace, have been chosen as representative examples of both technologies, and are presented below.

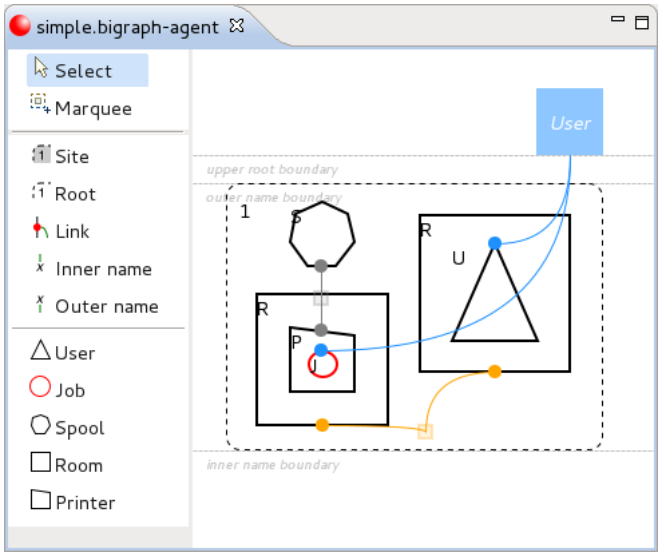


Figure 7: Big Red's bigraph editor.

Big Red's bigraph editor, shown in Fig. 7, is essentially a structured vector graphics editor. The palette on the left provides the tools that can be used to modify the model—for example, the 'Link' tool can be used to connect points and links, and the 'Room' tool can be used to create new nodes whose control is `Room`—while the view on the right shows the view of the model. (The bigraph editor also integrates with Eclipse's 'Outline' view, using it to display the place graph.)

The editor enforces the structural rules and visual conventions of bigraphs; if the user attempts to modify the model in a way that would cause them to be violated (for example, by dragging node `J` out of root `1`), an explanatory error message will be displayed. It can also (optionally) provide snap-to-grid and snap-to-object editing behaviour, as well as guidelines indicating the visual conventions (these are visible in Fig. 7).

The signature editor, shown in Fig. 8, manages both the bigraphical and visual properties of controls. The list view on the left shows the controls of the signature, and allows controls to be added and removed, while the panel on the right allows controls to be modified.

The appearance and arity of a control are defined using the canvas in the centre-right of Fig. 8.

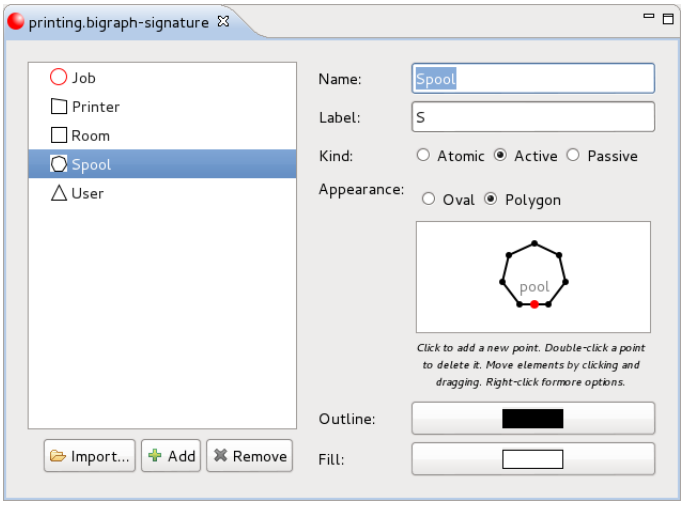


Figure 8: Big Red's signature editor.

The black circles are the vertices of the control's polygon, while ports are shown on the polygon as red circles (in this case, only one is shown). All of these objects can be created, moved and removed using the canvas and its context menu.

Visual fallback behaviour. Although Big Red is reliant upon layout information, the user interface can also load model objects that lack it; in these cases, automatically-generated layout information will be assigned.

4.3 The bigraphical model

Big Red's implementation of the bigraphical model strikes a balance between theoretical purity and practicality:

- it provides a pure, concrete bigraphical model, which can be extended with new properties and new constraints to implement other variations of bigraphs;
- the model can only be modified by executing an *edit script* (in the style of Højsgaard [Høj12]), a reversible description of a modification, which allows related changes to be grouped together and validated as a whole (and which provides reliable and automatic undo and redo support);
- it defines a standard XML representation of the model objects (and schemas to validate them), which can be extended to portably include new properties and relations; and
- it is independent of Eclipse, so external tools can use it without pulling in hundreds of megabytes of dependencies—although a separate Eclipse integration component makes it easy to use from within the Eclipse environment.

Some of the extensions built on top of the model are presented below.

Visual information. Big Red uses the model's extensibility mechanisms to implement large parts of its own functionality—the visual information used to draw, colour, and position objects, for example, has no privileged support.

Parameterised controls. Consider a simple extension to the bigraphical model: *parameterised controls*. These are identical in every way to traditional controls, except that they can optionally specify a *type* (for example, 'string' or 'integer'); in turn, a node whose control specifies a type can specify a *parameter* of that type. (Parameterised controls provide a convenient and simple way to introduce infinitely many controls; however, note that any concrete bigraph will contain only finitely many controls.)

An implementation of this extension (consisting of around 250 lines of Java code and 20 lines of XML) is included with Big Red. Controls and nodes are given "type" and "parameter" properties, support for which is also added to the file formats, and a node's parameter can only be changed in accordance with its control's type.

While this is a simple extension to the *model*, it could serve as a foundation for more interesting work elsewhere—for example, a tool built to use this model could assign some semantics to parameters in order to cleanly implement extra-bigraphical operations on data (like arithmetic or counters).

4.4 Interacting with External Tools

Big Red defines a special extension point for plugins that want to operate on a complete bigraphical reactive system. When an extension registered with this point is activated, Big Red's user interface is suspended, and the extension takes over. This extension point is a simple way of adding new functionality to Big Red: it essentially gives developers the ability to write small subprograms that operate on Big Red's model objects without having to delve too deeply into the workings of Eclipse.

The integration between Big Red and BigMC [PDH12] is implemented in this way—as an plugin which converts a Big Red model into its BigMC term language representation, executes BigMC as a subprocess, and parses the results back into Big Red model objects so that they can be visualised.

5 Conclusion

We have presented a brief introduction to the bigraphical formalism, outlined the architecture and design of Big Red, and described the motivation for developing the tool. Big Red and its accompanying user documentation are available from http://bigraph.org under the terms of the Eclipse Public License.

Big Red is still under active development. We intend to integrate support for other bigraph tools, and—together with the University of Udine—we are working on developing Big Red into a generally-useful platform for building and hosting new tools for bigraphs. In particular, we intend to use its model as an interchange format, unifying the previously incompatible and tool-specific representations of bigraphical reactive systems.

6 Acknowledgements

This research is supported by IT University of Copenhagen and the Danish Council for Strategic Research, grant no.: 2106-080046.

Bibliography

[BDE+06] *Bigraphical models of context-aware systems*. 2006. doi:10.1007/11690634_13

[CS12] M. Calder, M. Sevegnani. Process algebra for event-driven runtime verification: a case study of wireless network management. In *Integrated Formal Methods*. Pp. 21–23. 2012.

[DK08] T. Damgaard, J. Krivine. A generic language for biological systems based on bigraphs. Technical report, Citeseer, 2008.

[GDBH07] A. Glenstrup, T. Damgaard, L. Birkedal, E. Højsgaard. An implementation of bigraph matching. 2007.

[Gre09] C. Greenhalgh. bigraphspace. 2009. http://bigraphspace.svn.sourceforge.net/

[HNO06] T. Hildebrandt, H. Niss, M. Olsen. Formalising Business Process Execution with Bigraphs and Reactive XML. In *COORDINATION'06*. Lecture Notes in Computer Science 4038, pp. 113–129. Springer-Verlag, Jan. 2006. doi:10.1007/11767954_8

[Høj12] E. Højsgaard. *Bigraphical Languages and their Simulation*. PhD thesis, IT-Universitetet i København, 2012.

[Jen06] O. Jensen. Mobile Processes in Bigraphs. October 2006. Available at http://www.cl.cam.ac.uk/~rm135/Jensen-monograph.pdf.

[LM06] J. Leifer, R. Milner. Transition systems, link graphs and Petri nets. *Journal of Mathematical Structures in Computer Science* 16(6):989–1047, 2006. doi:10.1017/S0960129506005664

[Mil06] R. Milner. Pure Bigraphs: Structure and Dynamics. *Information and Computation* 204(1):60–122, Jan. 2006. doi:10.1016/j.ic.2005.07.003

[Mil09] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[PDH12] G. Perrone, S. Debois, T. Hildebrandt. A Model Checker for Bigraphs. In *ACM Symposium on Applied Computing 2012 – Software Verification and Testing Track*. ACM, 2012.