

# Practical Modelling with Bigraphs

BLAIR ARCHIBALD, University of Glasgow, UK

MUFFY CALDER, University of Glasgow, UK

MICHELE SEVEGNANI, University of Glasgow, UK

Bigraphs are a versatile modelling formalism that allows easy expression of placement and connectivity relations in a graphical format. System evolution is user defined as a set of rewrite rules. This paper presents a practical, yet detailed guide to developing, executing, and reasoning about bigraph models, including recent extensions such as parameterised, instantaneous, prioritised and conditional rules, and probabilistic and stochastic rewriting.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Models of computation**; • **Computing methodologies** → **Modeling and simulation**.

Additional Key Words and Phrases: Bigraphs, Graph Rewriting, Rewrite Systems

## ACM Reference Format:

Blair Archibald, Muffy Calder, and Michele Sevegnani. 2025. Practical Modelling with Bigraphs. 1, 1 (February 2025), 37 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

Bigraphs are a universal modelling formalism for describing systems that evolve in space, time, and connectivity. They were introduced by Milner [32, 34], and have been extended to directed, stochastic, sharing, conditional, and probabilistic bigraphs [3, 4, 21, 27, 39]. While they have seen use in modelling a range of systems including: mixed-reality games [7], network management [10], wireless communication protocols [11], biological processes [27], cyber-physical security [2, 42], indoor environments [43], and sensor systems [41], they have not yet seen widespread adoption. One reason is the early emphasis on theoretical aspects, including the relationship to specific mathematical categories, and tools for deriving bisimulation congruences that are common in work on process calculi. Less attention was given to bigraphs for system modelling and analysis. The purpose of this paper is to provide practical guidance on how to model with bigraphs and to illustrate some of our extensions to bigraph theory that enhance modelling in practice.

A bigraph consists of a set of user defined *entity types* relevant to the domain being modelled, *e.g.* Computer, Person, Room, Cell, Protein, ..., which can be related both *spatially* through nesting, *e.g.* a Person *in* a Room, or and through *linking*, *e.g.* communication between Computers in different Rooms. Spatial relations are described by *place graphs* (a forest), with *regions* indicating modules, or adjacent parts of the system; linking is described by *link graphs* (a hypergraph). A bigraph *reactive system* (BRS) consists of bigraphs and user-defined *rewrite rules* that define how bigraphs evolve

---

Authors' addresses: Blair Archibald, University of Glasgow, UK, [Blair.Archibald@glasgow.ac.uk](mailto:Blair.Archibald@glasgow.ac.uk); Muffy Calder, University of Glasgow, UK, [Muffy.Calder@glasgow.ac.uk](mailto:Muffy.Calder@glasgow.ac.uk); Michele Sevegnani, University of Glasgow, UK, [Michele.Sevegnani@glasgow.ac.uk](mailto:Michele.Sevegnani@glasgow.ac.uk).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2025 Association for Computing Machinery.

XXXX-XXXX/2025/2-ART \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

over time. For example, the rules might express circumstances under which a Person leaves a Room or a Computer is connected/disconnected to a Network.

A core advantage of bigraphs over other modelling formalisms is the diagrammatic notation, backed by an *equivalent* algebraic form, which provides intuitive descriptions of systems without requiring detailed knowledge of mathematical description languages. The notation is expressive, compared with similar diagrammatic formalisms such as Petri-nets [18], as entity types (including connectivity, placement etc.), their diagrammatic representations, and the rewrite rules for updating a model, are all *user* defined. We have found the diagrammatic notation particularly useful and accessible to system designers and users when developing models.

The example in Fig. 1 illustrates a bigraph model, in diagrammatic format. In Fig. 1a there are two regions (the dashed rectangles), indicating there are two distinct parts of the model: physical and data. The physical region consists of a Room, containing a Wi-Fi-enabled Display, a User, and their Phone, which is connected to the Display. The data region consists of the User's Name and Address, which consists of a Street and House. In general, we may use (coloured) shapes to denote different entities. Links are in green and may be named, *e.g.*  $r$ ,  $w$ , which indicates a potential link to other entities. Fig. 1d contains an example rewrite rule: a User leaves a Room, taking their Phone with them. The Display may be connected to other devices (*e.g.* more phones or computers), which is indicated, on the left-hand side, by the link named  $w$ . On the right-hand side, the User and their Phone are no longer in the Room and the Display and Phone are disconnected, but any other connections  $w$  remain (including no connection). The gray rectangles denote that anything else that may have been in the Room will remain in the Room unchanged. Fig. 1e shows the transition that results when we apply this rewrite rule to our example bigraph. Notice that the data region is unchanged.

Our aim is to provide a core intuition behind bigraphs and familiarity with the diagrammatic and text formats. To this end, we avoid giving formal mathematical descriptions where possible and focus on example-driven explanations. We do not assume any existing knowledge, and by the end of this paper a reader should be able to create their own bigraph models and use them for rigorous system design and analysis. A previous example-driven tutorial [17] showed the versatility of bigraphs for modelling, however it does not account for recent innovations, *e.g.* parameters, priorities, and conditional bigraphs [3], nor give detailed descriptions of modelling techniques/styles that are common in practice, *e.g.* how to apply a rewrite rule a certain number of times, or how to model multiple perspectives. Likewise, existing bigraph publications usually show models without explaining in detail why these modelling decisions were made.

To show bigraphs are not just a theoretical tool, but a practical one, we provide fully executable model files of each example [6] in BigraphER format, and a wider collection of examples is available online<sup>1</sup>. BigraphER [40] is an open-source framework for manipulating, visualising, and executing bigraphical reactive systems by applying rewrites. We choose BigraphER as it is actively maintained, features an intuitive syntax similar to the algebraic definition of bigraphs, and has a range of state-of-the-art extensions such as probabilistic rewriting [4] and conditionals [3]. Examples not requiring these features are applicable to other bigraph tools such as JLibBig [13], which also supports directed bigraphs (we discuss emulating directed links in Section 7.2), or the Bigraph Toolkit Suite [22]. Historical tools such as BPL [26] (Bigraphical Programming Language), BigMC [37] (Bigraph Model Checker), and BigRed [20] are unmaintained.

<sup>1</sup><https://uog-bigraph.bitbucket.io/examples.html>

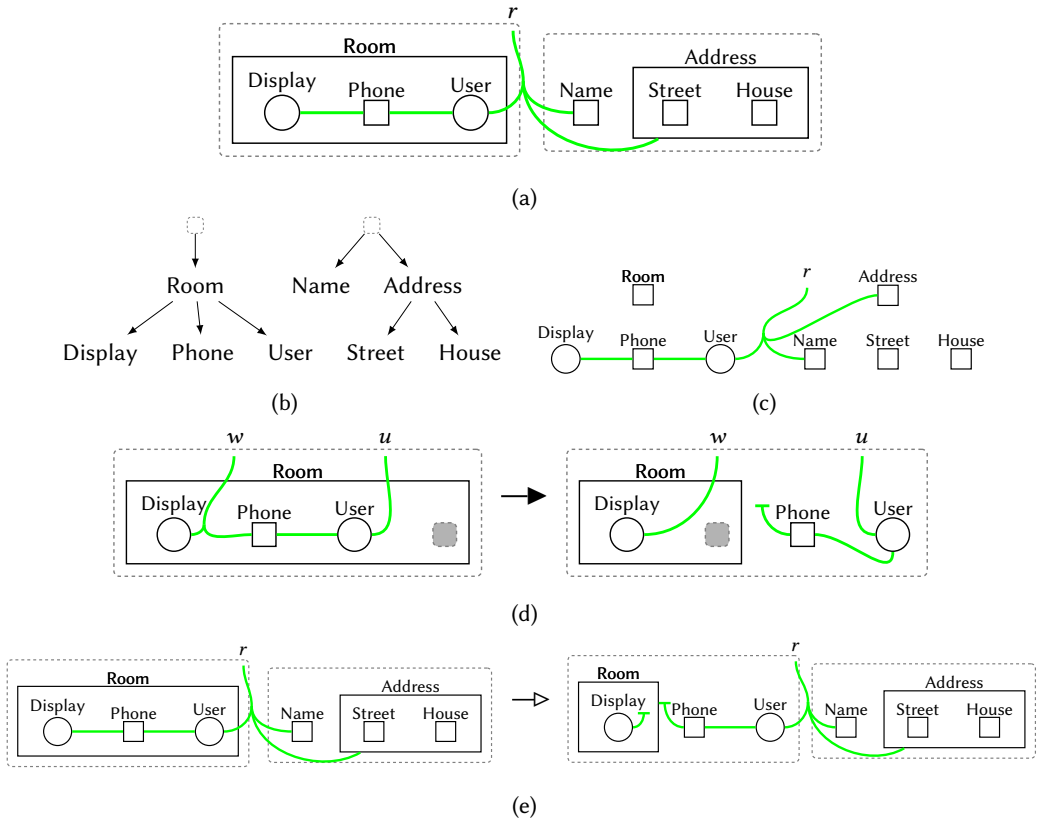


Fig. 1. Example bigraph with two regions: a physical Room containing a Wi-Fi-enabled Display, a User, and their Phone, and data consisting of a Name and an Address. (a) Diagrammatic representation: entities are black shapes and links are green lines. (b) The place graph. (c) The link graph. (d) Example reaction rule: a user leaves a room and takes the phone with them. (e) Application of the reaction rule from (d) to (a).

### 1.1 Paper organisation

The next five sections cover the basics of bigraphs and BigraphER. In Section 2 we show how place graphs are used to model topological relationships, *e.g.* the spatial arrangement of entities; in Section 3 we show how link graphs are used to model linking relationships such as communication; and in Section 4, we show to model bigraph evolution through (user-defined) rewrite rules. Section 5 introduces a number of extensions we have defined and implemented: parameterised entities and rules, and instantaneous and conditional rules.

The following three sections offer practical modelling advice for common scenarios: multi-perspective modelling in Section 6, entity versus link structures in Section 7, and rewriting control in Section 8, including phases and turn taking.

Further rewriting extensions: probabilistic, stochastic, and non-deterministic rewriting, are covered in Section 9, and in Section 10 we give an overview of analysis through state space exploration both using simulation (single trace) and model-checking (sets of traces). We also introduce bigraph patterns and give pointers to several detailed bigraph models and their analysis. In Section 11 we comment on the types of problems we think are best suited to Bigraphs and we

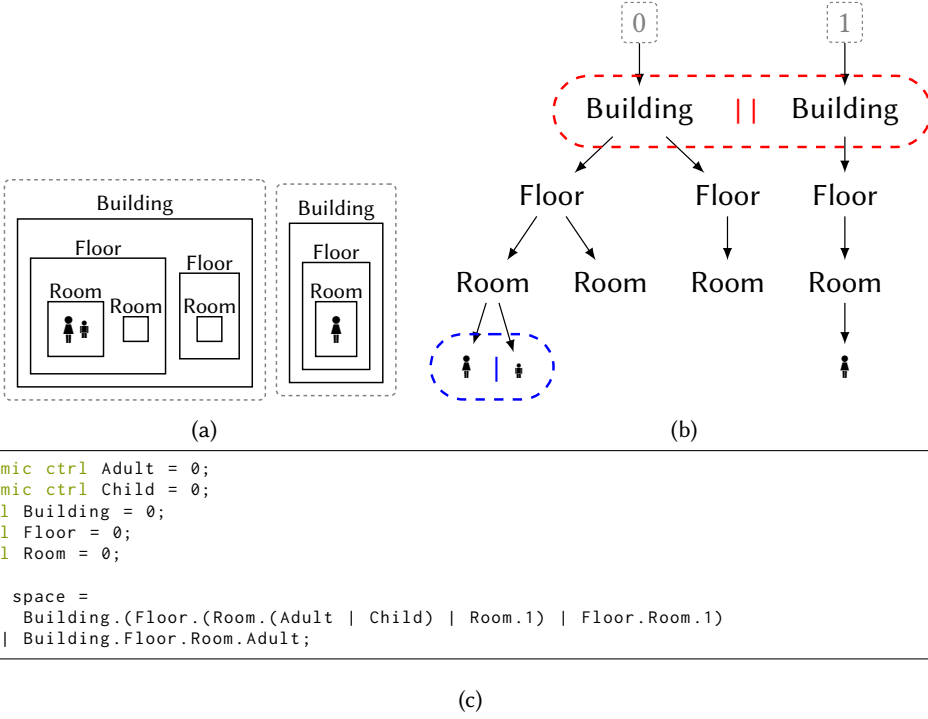


Fig. 2. Modelling buildings. (a) Diagrammatic place graph (b) Forest representation (c) BigraphER model. In (b), the red and blue dashed ovals, containing red and blue parallel and merge product operators resp., are superposed on the place graph. These are not part of bigraph notation, but serve to highlight the difference between  $||$  and  $|$ .

conclude in Section 12. Modelling tips are given throughout and summarised in Appendix A. A note on port ordering in abstract bigraphs is contained in Appendix B.

## 2 PLACE GRAPHS

The place graph describes the *topological* relationships between entities. It is used to model spaces, for example location: an *Adult* is in a *Room*, or ownership: a *BatteryLevel* belongs to a specific *Phone*. In standard bigraphs, these relations are described by *forests* that are a collection of *trees*, *i.e.* you are allowed multiple roots. In Section 2.4 we consider an extension that allows directed acyclic graphs instead of forests.

To illustrate place graphs, we show a model of locations and people (adults and children) within a building in Fig. 2. We give three equivalent representations: Fig. 2a shows the standard way to draw a place graph where *nesting* is explicit, Fig. 2b shows the equivalent place graph as the underlying forest, and Fig. 2c shows the BigraphER code corresponding to this graph. The binary operator  $.$  is used to indicate nesting. For example, *Room.Adult* means an *Adult* within a *Room*. The keyword *atomic* indicates an entity that cannot contain any children.

Types are defined with keyword *ctrl*<sup>2</sup>. Keyword *big* introduces a named bigraph. BigraphER comments are indicated by  $\#$ . Entities are allowed any number of children—including none which is represented by the special bigraph  $1$ —and siblings are defined using *merge product* written  $|$ .

<sup>2</sup>We prefer the reserved word *type* but have remained faithful to Milner’s reference to *control*.

Note that each entity also has “ = 0”. We have included this so that the BigraphER code is correctly formed; strictly, this refers to the link graph, which will be explained in Section 3.

For example, `Room.(Adult | Child)` is a room containing one adult and one child. Merge product is *commutative*, meaning children are *unordered*. That is, `Room.(Adult | Child)` and `Room.(Child | Adult)` model the same room; if ordering is required it must be explicitly encoded (see Section 7.3).

To create multiple *regions* (roots) we have the operator `||` called *parallel product*. Parallel product can be seen as the juxtaposition of regions. For example, `Building || Building` is the forest with two trees, each containing a building. Unlike merge product, parallel product is not commutative. However, when used in reaction rules (see Section 4) the ordering does not matter.

**Modelling Tip 1:** `||` and `|` allow us to build bigger bigraphs from smaller. Use `||` to model distinct bigraphs and `|` for merging bigraphs.

Notice that entities are typed, but do not have names, e.g. there may be several `Room` entities in a `Floor` but we cannot identify a specific one. Formally, bigraphs without identifiers are called *abstract bigraphs* whereas *concrete bigraphs* assign identifiers to entities. In this paper we refer exclusively to abstract bigraphs.

## 2.1 Regions and sites

Bigraphs are *always* rooted<sup>3</sup> using *regions*, represented by the dashed rectangles. Regions indicate adjacent parts of the system. In the buildings example there are two regions, one for each building.

We stated above that the place graph `1` indicates an entity has no children; now we can be more precise—it represents the place graph with a single region and nothing else.

We can abstract away from entities using *sites*, which are like variables (see Section 4.1). We draw them as dashed gray filled rectangles. For example, in Fig. 3, each `Room` contains one site that represents one or more bigraphs. A site can be nested wherever an entity can be nested, including directly under a region, in which case this region/site pair is called the *identity* place graph and denoted by `id`.

An example containing regions and sites is in Fig. 3, where we focus on a single floor of a building, having abstracted away the specific contents of rooms using sites.

## 2.2 Two special place graphs: `id` and `1`

We use `1` with a *non-atomic* entity to indicate that sites are not required, for example, when a room is empty, as indicated in line 8 of Fig. 2c. Notice that in the diagrammatic form, e.g. in Fig. 2a, we simply do not draw anything underneath `Room` to indicate an empty room, whereas in the textual form, we have to make this explicit with the text `1` (as an operand of the nesting operator `.`).

To summarise the difference between `1` and `id`, consider Fig.’s 2 and 3. `Room.1` indicates a room with no possibility of children, whereas `Room.id` indicates a room with a site, which may be instantiated with *zero* or more (children). That is, sites might themselves contain `1`, meaning there is nothing inside them.

**Modelling Tip 2:** Use `1` to indicate “no possibility of any children” and `id` to indicate “zero or more children”.

<sup>3</sup>This only applies to standard bigraphs. Bigraphs with sharing allows bigraphs with no parents.

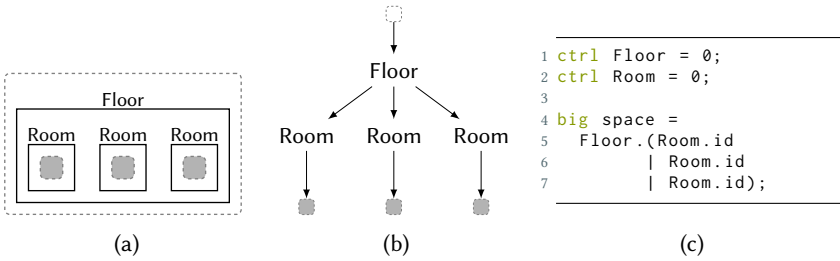


Fig. 3. Place graph with one region and three sites.

### 2.3 Diagrammatic notation

The shapes and colours of entities in the diagrammatic<sup>4</sup> notation are chosen by the user. In this paper we mainly use simple geometric shapes that are easy to draw such as square, rectangle, and circle, but any shape is possible, as well as colour or shading.

Shading might be used to represent a combination of bigraphs that denotes an entity in different states or stages of a process, e.g. we might use a circle for `File` and a red circle for `File.Open` or alternatively to represent a new entity type `FileOpen`. Note that in the theory there is no subtyping so `File` and `FileOpen` are distinct entities. By giving them the same shape we are giving (informally) an additional relationship between these entities.

**Modelling Tip 3:** Use prefixes/suffixes in entity names (in textual format) and colours and shading (in diagrammatic format) to indicate relationships between states or stages of a process.

### 2.4 Sharing

In the standard definition of bigraphs, each entity is allowed only a single parent (another entity or a region). However, it may be natural to allow a single entity to have multiple parents, for example when modelling spatial overlap such as wireless signal ranges or fields of vision.

*Bigraphs with Sharing* [39] relax the restriction on place graphs from a forest to a directed acyclic graph, meaning entities can have any number of parents (including being in multiple regions; or having no parents at all). While there are a few key theoretical implications, from a modelling viewpoint, bigraphs with sharing are a simple extension.

We illustrate by introducing security `Cameras` into our building model. A room may have multiple security cameras, each of whose field of vision might overlap, meaning that a single `Adult` entity might be nested under two (or more) cameras at a time. This is shown in Fig. 4c, where line 8 by  $([\{0, 1\}, \{1\}], 2)$  specifies a map  $[r_0 \mapsto \dots, r_1 \mapsto \dots, \dots]$  of regions to sets of sites. Here it indicates the first region (containing `Adult`) should appear in both sites  $\{0, 1\}$  while the second region (containing `Child`) should only appear in site  $\{1\}$ . We show this diagrammatically in Fig. 4d. The additional 2 is required as the mapping might not be surjective, e.g. we may choose to ignore a region of the `share` bigraph.

## 3 LINK GRAPHS

Bigraphs allow us to express non-spatial connectivity through the *link graph*, which is a set of *hyperedges*. Each hyperedge consists of a non-empty set of vertices. This contrasts with conventional graph edges that define one-to-one relationships (i.e. a hypergraph where all sets have cardinality two).

<sup>4</sup>This is sometimes referred to as graphical notation, which we avoid in case of confusion with place and link graphs.

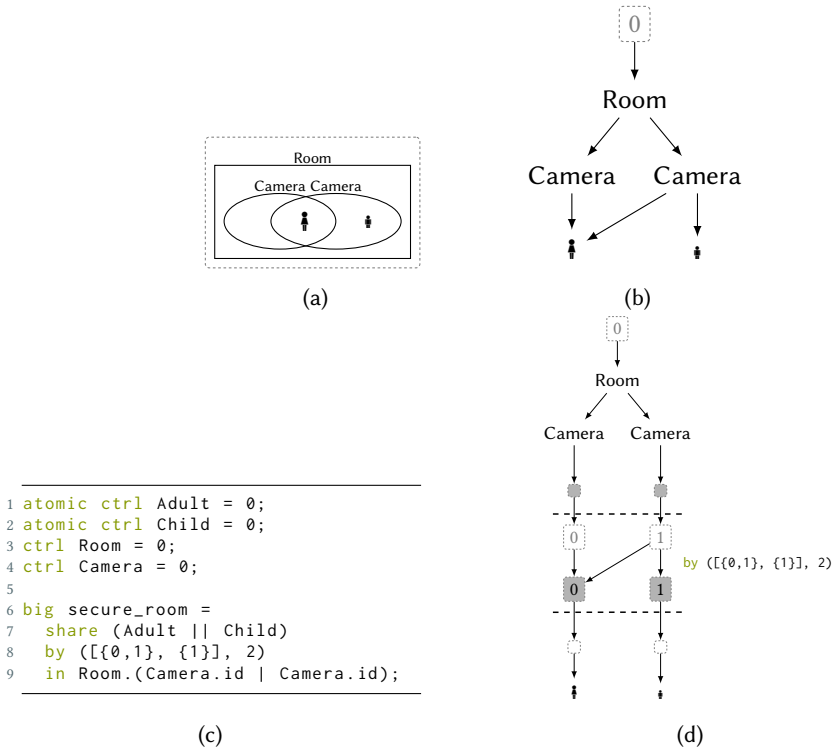


Fig. 4. Bigraphs with sharing: cameras with overlapping fields of vision that may capture a single Adult. (a) Bigraph model. (b) Place graph. (c) BigraphER snippet. (d) Deconstructed place graph showing how `by` expressions work.

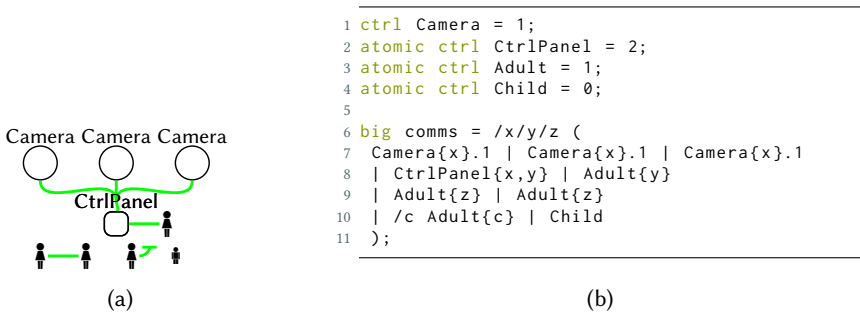


Fig. 5. Link graph example: CCTV, phone calls, and remote network access. (a) Link graph. (b) BigraphER snippet.

Each entity involved has a **fixed** *arity* that defines the number of *ports*: a name that reflects their role in communication. In the diagrammatic form, ports are implicit—the point at which the edge meets the entity boundary. In the text form, *i.e.* BigraphER snippet, entity definitions include their arities following the symbol `=`. For example *e.g.* `atomic ctrl Camera = 1`, specifies a `Camera` entity has 1 port.

A link may be *disconnected*, *i.e.* the entity is not linked to any other entity, (a one-to-zero hyperedge—a singleton set). This is drawn as a single edge with a short perpendicular line at one end (not the entity).

To illustrate link graphs, in Fig. 5a we have security cameras that can be linked to form CCTV, and adults that can communicate via phones or dial into the security control panel. Links are drawn in green (any colour can be used). All Camera entities have arity 1 and are connected via a single hyperlink to the security control panel. Adult entities also have arity 1 allowing them to communicate, either with other people or via remote access to the security control panel. One of the Adults (third from left) is not connected anywhere and so their link disconnected; the single Child entity has no ports. Note, all entities appear in the link graph, regardless of whether they have any links.

### 3.1 Closed and open links

A link may be partially specified, in which case it is *named* to indicate it *may* have connections elsewhere. An example of this is the link named *w* in Fig. 1, which indicates the display may link to other devices. This use of a name is similar to a free variable and we call such a link *open*. The alternative is a *closed* link, which is fully specified, *e.g.* the link in Fig. 1 between the Phone and User.

In the text form of bigraphs we use identifiers *e.g.* *x*, *y* to define both links and names, the idea being that ports that share an identifier share a link. An example is in BigraphER code of Fig. 5b. The *closure* operator */* indicates the link that is identified is closed. For example, *e.g.* the link identified by *x* is bound in */x* (`Device{x} | Device {x}`) and so the link is closed and cannot connect elsewhere. Like bound variables in programming languages, the actual identifier does not matter, and so */y* (`Device{y} | Device{y}`) is an equivalent bigraph. This also means **there is no global way to refer to a specific link**, *i.e.* identifiers do not identify specific edges and so you cannot refer to, for example, “the edge named *y*”. When an identifier is not bound it becomes a *name*, allowing the link it defines to be extended. Finally, a name is *idle* when it exists but is not connected to any other entities or names.

We illustrate links and names in Table 1, which is based on the declaration `atomic ctrl Device = 1` that indicates a device has no children and one port.

**Modelling Tip 4:** Use a named, open link to indicate “this link *potentially* connects elsewhere”, and a closed link to indicate “*only* these entities are connected”. The specific names used do not matter.

### 3.2 Bigraphs: Combining Place and Link Graphs

A bigraph consists of a place graph (Section 2) and a link graph (Section 3) defined over the same set of entities. A summary of the main components of bigraphs is in Table 2.

An example bigraph bringing together place and link graphs is in Fig. 6. When drawing bigraphs we overlay the link structure on the place structure. By convention, open link names are always drawn above a bigraph. Finally, we introduce additional terminology that allows us to ensure correct bigraph composition and rewriting: link graphs have *inner* and *outer* names. Outer names are the open links. Inner names occur rarely when developing a bigraph as a model<sup>5</sup>, and do not occur in any examples in this paper; inner names exist mainly in internal bigraphs during rewriting.

<sup>5</sup> A normal form for bigraphs [34, Chapter 3] allows most linking to be pushed upwards and joined at the top-level and this allows most inner names to be expressed as outer names instead.

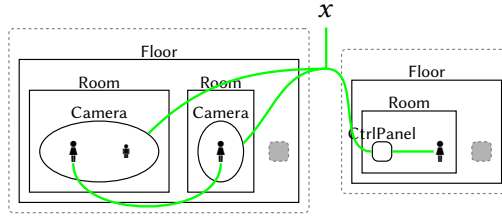


Table 1. Links and names: BigraphER notation on the left, diagrammatic notation on the right (a) A closed link between two connected devices, no other connections are possible. (b) Two connected devices with a further potential connection named  $x$ . (c) A disconnected device. (d) A device with a potential link  $x$ . (e) A closed link (to the left) and an open link (to the right),  $x$  is bound in the former and free in the latter. (f) An idle link  $x$ .

(a)	$/x \text{ (Device } \{x\} \mid \text{Device}\{x\})$	
(b)	$\text{Device}\{x\} \mid \text{Device}\{x\}$	
(c)	$/x \text{ Device}\{x\}$	
(d)	$\text{Device}\{x\}$	
(e)	$/x \text{ (Device}\{x\} \mid \text{Device}\{x\}) \mid \text{Device}\{x\}$	
(f)	$\{x\}$	

Table 2. Main Bigraph elements and syntax.

Element	BigraphER Syntax	Diagram
Entity of arity 1	$K\{a\}$	
Name closure	$/a K\{a\}$	
Identity Place Graph	$\text{id}$	
Identity Link Graph	$\text{id}\{x\}$	
Empty Region	$1$	
Idle (outer) name	$\{x\}$	$x$
Nesting	$/x A\{x\}.B\{x\}.\text{id}$	
Parallel product	$C\{x\}.\text{id} \parallel D\{x\}.\text{id}$	
Merge product	$C\{x\}.\text{id} \mid D\{x\}.\text{id}$	



(a) Bigraph model of a building.

---

```

1 atomic ctrl Adult = 1;
2 atomic ctrl Child = 0;
3 ctrl Floor = 0;
4 ctrl Room = 0;
5 ctrl Camera = 1;
6 atomic ctrl CtrlPanel = 2;
7
8 big space =
9   Floor.(/y (Room.(Camera{x}.(Adult{y} | Child))
10    | Room.(Camera{x}.Adult{y})) | id)
11 || Floor.(Room.(/z (CtrlPanel{x,z} | Adult{z})) | id);

```

---

(b)

Fig. 6. Example bigraph model of a building. (a) Bigraph representation (b) BigraphER snippet. There is one open link named  $x$ .

We note there is a similar concept for place graphs, but does not require additional terminology: *sites* are *inner* and roots are *outer*.

#### 4 BIGRAPHICAL REACTIVE SYSTEMS

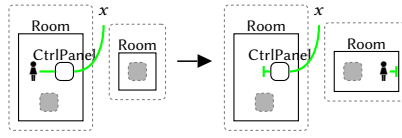
To encode system dynamics, *i.e.* how bigraphs *evolve*, we require a set of reaction rules (also called rewrite rules). A distinctive feature of bigraphs as a modelling tool is that the rules are *user defined*. Bigraphs together with rewrite rules are known as a Bigraphical Reactive System (BRS).

Reaction rules have form  $L \rightarrow R$ , where  $L$  and  $R$  are bigraphs; the rule specifies that an instance of  $L$  can be substituted by  $R$ . To apply a rewrite rule, we find an occurrence of sub-bigraph  $L$  within a larger bigraph  $B$  and substitute  $L$  with  $R$  creating a new larger bigraph (state)  $B'$ . The relation induced by the rewrite rules is denoted by  $\rightarrow$ ; *i.e.*  $B \rightarrow B'$  if  $B$  can rewrite to  $B'$  by application of a rule. While in general  $L$  and  $R$  can be any bigraphs, **the interfaces (sites/regions/names) of  $L$  and  $R$  must be equal.**

When more than one reaction rule applies, or the same reaction rule has multiple occurrences, then the rule applies non-deterministically, *i.e.* we pick any and apply it.

An example reaction rule, `leave_secure`, is in Fig. 7a, with BigraphER representation in Fig. 7b. In BigraphER we write rules with `-->` separating  $L$  and  $R$ . The rule describes how a Person, who is connected to a security `CtrlPanel`, can move between rooms. Importantly, when moving out of the room containing the `CtrlPanel`, they sever their link to the `CtrlPanel` to avoid information leaks. Although parallel product is not commutative, for *matching*, the order we specify the regions does not matter<sup>6</sup>, *i.e.* this will find *any* two rooms not only those where the second room is to-the-right in the larger bigraph. The result of applying `leave_secure` to a given bigraph is in Fig. 8. Note there is only one valid occurrence for `leave_secure`, and so all other entities in the system are unchanged during the rewrite.

<sup>6</sup>Any required explicit symmetry operations are added during decomposition.



(a)

```

1 atomic ctrl Person = 1;
2 atomic ctrl CtrlPanel = 2;
3 ctrl Room = 0;
4 ctrl Floor = 0;
5
6 react leave_secure =
7 /z Room.(Person{z} | CtrlPanel{z,x} | id)
8 || Room.id
9 -->
10 Room.(/z CtrlPanel{z,x} | id)
11 || Room.(/z Person{z} | id);
12
13 big initialBigraph =
14 /z Floor.(
15     Room.(Person{z} | /x (Person{x} | /y CtrlPanel{x,y}))
16     | Room.Person{z}
17 );
18
19 begin brs
20   init initialBigraph;
21   rules = [ {leave_secure} ];
22 end
    
```

(b)

Fig. 7. BRS with a single reaction rule `leave_secure`. (a) Graphical notation for `leave_secure`. (b) BigraphER snippet.

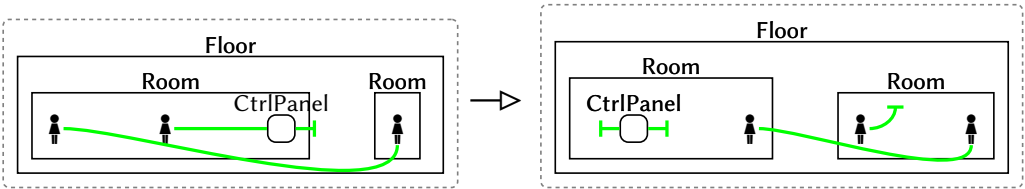


Fig. 8. Applying `leave_secure` to a simple scenario.

Sites/regions/names are especially important for rewriting as they allow a single rule to be applied in many circumstances. In this case, the two sites allow any other bigraph to exist (including the empty bigraph) within the rooms, and intuitively we can think of the rule as saying “find a room with *at least* one Person and one CtrlPanel who are linked, and another Room that contains anything, including nothing”. The use of parallel product (two *regions*) means the two Rooms cannot be a descendant of the other and do not need to be siblings (but can be), *i.e.* the Person could move to a Room on a different floor, or even a different building. The name *x* allows the CtrlPanel to (possibly) be connected elsewhere, *e.g.* to cameras in the building. If the link was closed (unnamed) the link must be an exact match.

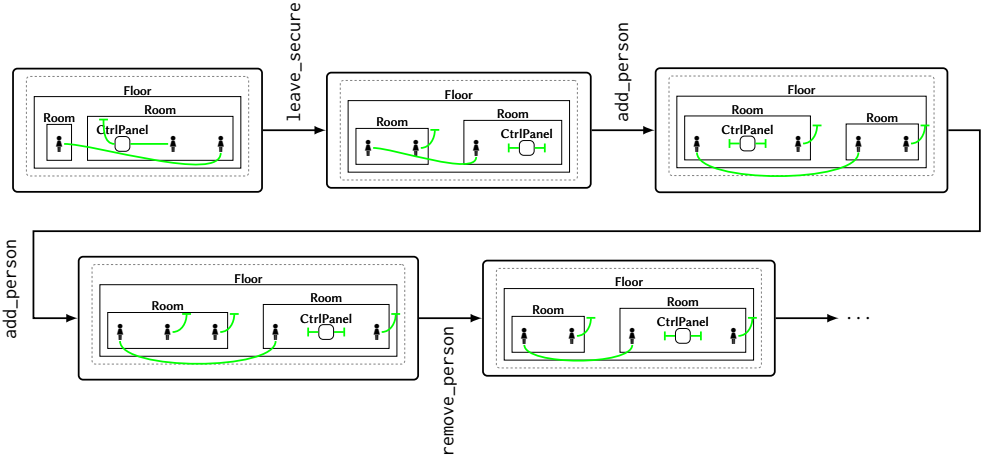


Fig. 9. Applying a sequence of rewrite rules to an initial state (top left).

Given a reaction rule, we define a BRS in BigraphER using a `begin brs ... end` block. Inside this block we specify an initial bigraph (initial state), using the syntax `init b` (where `b` is a named bigraph), and set of rules as shown in Fig. 7b.

To show execution of a model, we give a possible sequence of rewrites in Fig. 9 for a BRS with the rule `leave_secure` and two additional rules `add_person` and `remove_person` allowing people to enter or exit the system so long as they are not connected elsewhere. We use the same initial bigraph as in Fig. 7. This is one of many possible traces, e.g. we could also apply `add_person` in the first state. Notice that, as we use diagrammatic elements, we can move them around as we draw states so long as their relationships (nesting/linking) are maintained.

In practice, we require the left hand side of a rewrite to be *solid* [27]<sup>7</sup>.

A bigraph is *solid* if

- all regions contain at least one node and no outer names are idle
- no two sites or inner names are siblings
- no site has a region as a parent
- no outer name is linked to an inner name.

These constraints only apply to the left hand side. For example, we can disconnect a name, e.g.  $A\{x\} \dashrightarrow /y A\{y\} \mid \{x\}$ , resulting in an idle name on the right hand side ( $\{x\}$ ). BigraphER automatically enforces these constraints and we assume them throughout this paper.

#### 4.1 Sites as Variables: Manipulating Sites During Rewriting

Sites are a powerful feature of bigraphs because they act like variables. For example, consider a rule to delete data from a server as shown in Fig. 10. Without sites, we are forced to include rules to delete each number of data items as in Fig. 10a.

Sites allow us to specify a set of bigraphs without explicitly enumerating all the elements. That is, one rule can apply in many situations. For example, we can delete any number of data items (and anything else on the server) using a single rule shown in Fig. 10b. In this case, we have *deleted* a site and the entire bigraph it abstracted over is removed.

<sup>7</sup>This ensures unique occurrences, which is central to probabilistic and stochastic rewriting (Section 9).

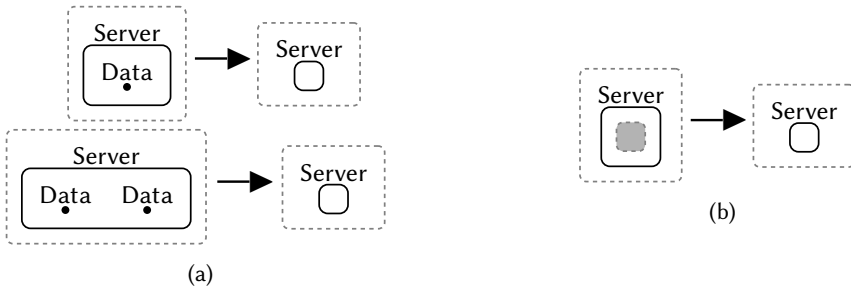


Fig. 10. (a) Delete requiring multiple rules. (b) Delete as a single rule using a site.

**Modelling Tip 5:** Sites are abstractions over bigraphs, i.e. they are bigraph variables that can be instantiated with a bigraph. Sites should be used when defining general rules that apply in many situations.

During rewriting we can *duplicate* or *discard* the contents of a site using a special construct known as an *instantiation map*. These operations occur frequently in practical bigraph models.

We identify sites numerically based on their position in a rule definition, with the left-most site being site 0. For example in  $A.id \ || \ B.id$  the site below A is site 0 and below B is site 1. An instantiation map determines, for each site in the right-hand-side of a rule, which sites this maps to on the left-hand-side of a rule. For example, an instantiation map:  $0 \mapsto 1, 1 \mapsto 0$ , gives site 0 on the right, the contents of site 1 on the left, and site 1 on the right, the contents of site 0 on the left (i.e. it implements a swap). All sites on the right-hand-side must correspond to a site on the left, but not all left-hand sites need to be included. For example, the map  $0 \mapsto 0, 1 \mapsto 0$ , would duplicate site 0 from the left into both right-hand sites and the remaining site (site 1) on the left is discarded.

We write instantiation maps at the end of a rule definition using the syntax  $@[n, \dots, m]$  (for natural numbers  $n, \dots, m$ ). The value of the  $i^{th}$  element of this list determines the left-hand site that the  $i^{th}$  right-hand site corresponds to. For example  $@[0, 1, 1]$  maps sites  $0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1$ . This map must be fully defined and so the length should match the number of sites on the right hand side. This avoids the situation where we have a site but no information about how to instantiate it.

Graphically, we draw instantiation maps using blue dashed arrows. For clarity, we may sometimes draw only select arrows when it is a 1-to-1 mapping, other than specific sites, and it is clear from the rule what we intend.

We show the power of instantiation maps by example, including how they let us model *copy* and *delete* operations. Movement of entities happens often in physical scenarios, e.g. moving between rooms, but copying and deletion of entities is less common. To show these features of instantiation maps we extend our building example with servers and data.

An example rule using an instantiation map is in Fig. 11. In this case we *copy* everything that was in the database (including nothing if it was empty) to the local server, while keeping all local server data intact. To make this mapping clear we have numbered the sites in this example, but will not number sites in general. Similarly, by just changing the instantiation map, we model copy-and-delete in Fig. 12. In this case the site under the Database is not in the map and so is dropped.

**Modelling Tip 6:** Careful consideration needs to be given to duplicating sites when the bigraphs being duplicated contain links: when a site is duplicated that contains links, the links remain connected. For example, if we copy  $A\{x\}$  to obtain  $A\{x\} \ || \ A\{x\}$ , then both A entities are connected in the result<sup>8</sup>.

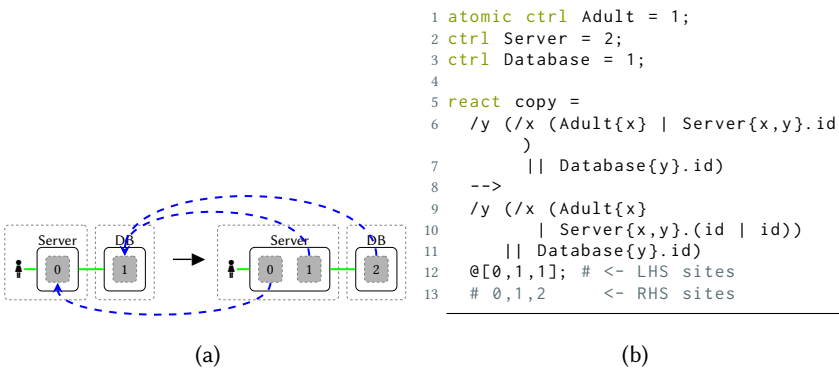


Fig. 11. Copy through instantiation map.

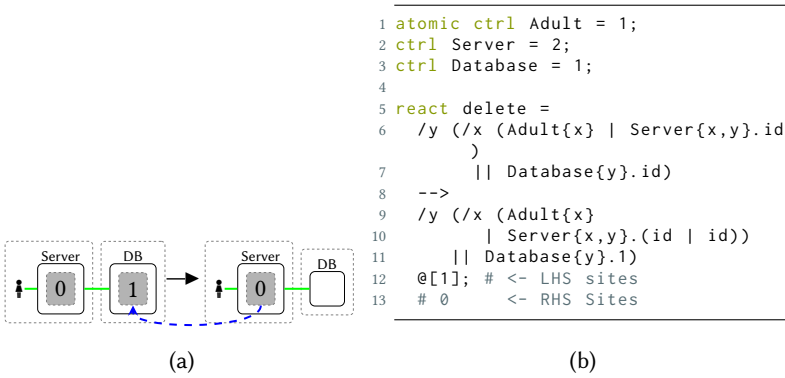


Fig. 12. Copy-and-Delete through an instantiation map.

## 5 PARAMETERISED, INSTANTANEOUS, AND CONDITIONAL RULES

We have extended bigraph entities and rewriting in several ways, all of which have been implemented in BigraphER.

### 5.1 Parameterised Entities

Models may require numeric operations or to assign identifiers to entities, *e.g.*  $\text{Person}(1)$ ,  $\text{Person}(2)$ , etc. While it is possible to encode numbers using schemes such as Peano arithmetic, these can be difficult to work with and have significant computational overhead.

BigraphER provides *parameterised entities*, *e.g.*  $\text{Nat}(n)$ ,  $n \in \mathbb{N}$  that represent *families* of entities, one for each value of  $n$ . We also support float and string parameters. We view this as syntactic sugar for defining a set of entities: *e.g.*  $\text{Nat}(\emptyset)$ ,  $\text{Nat}(1)$ ,  $\dots$ , where  $\text{Nat}(1)$  is just an entity in the same way  $\text{Server}$  is an entity. For practical models we choose  $n$  as finite (and defined by the user), although the theory supports infinite entity sets if required. Entities can vary in multiple parameters if required, *e.g.*  $A(n, m)$ . We use the syntax `fun ctrl A(x) = 0` (where  $x$  is an arbitrary identifier) to denote a parameterised entity<sup>9</sup>.

<sup>9</sup> `fun` is a reference to function, *i.e.* it takes parameters and produces new entities.

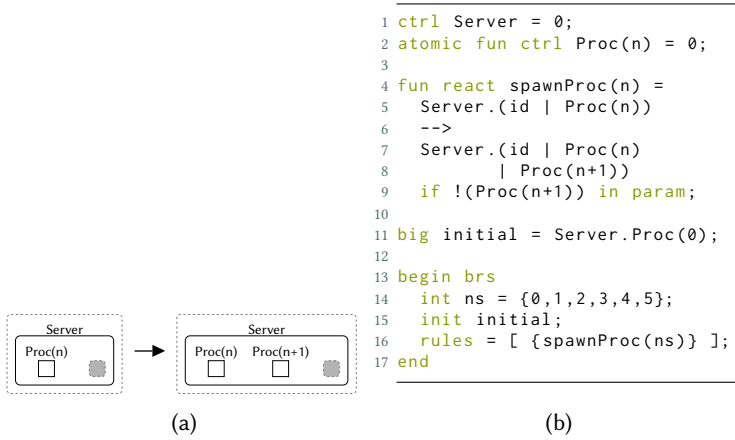


Fig. 13. Example parameterised rule. (a) Rule for spawning server processes. (b) BigraphER snippet.

### 5.2 Parameterised Rules

Parameterised rules represent a *family* of rules. For example, we can write  $r(n)$  for  $n \in \mathbb{N}$  in place of  $r(0), r(1), \dots$ . Within the rules we allow the bound variables to be used for parameterised entities.

As with parameterised entities, in practice parameterised rule variables must be instantiated with a finite set of values. In BigraphER we allow numeric operations on parameters to be applied within a rule. These operations are performed as the model is compiled, *i.e.* the resulting rules are fully instantiated.

A simple parameterised rule is in Fig. 13 where we allow processes running on a server to spawn future processes:  $\text{Proc}(n)$  spawns  $\text{Proc}(n+1)$ , up to  $n = 5$ . The restriction that  $\text{spawnProc}(n)$  is only specified for  $0 \leq n \leq 5$  is captured by the `rules` clause in line 16, in conjunction with the set `ns` specified on line 14. Currently there is no syntax to restrict parameter values at the point the rule is specified, and so it does not appear in the diagrammatic notation<sup>10</sup>.

In this case there is no rule `spawn_proc(6)`. We use the syntax `fun react` to define a parameterised rule instead of a standard rule.

**Modelling Tip 7:** Parameterised rules are syntactic sugar for a set of underlying rules, so use them sparingly as each new rule increases the work needed for system analysis. In practice, this affects how you describe entities. For example, it is often better to define `Camera.CName(1)` which can be abstracted by a site whenever the identifier is unimportant, *e.g.* `Camera.id`, rather than `Camera(1)`, which requires a family of rules *every* time a rule uses a `Camera`.

We have extended bigraph rewriting in several ways, all of which have been implemented in BigraphER.

### 5.3 Rule priorities

In general a reaction rule can be applied whenever there is a suitable match. However in practice we often want more control over when rules can be applied. For example, consider the rule `leave_room` in Fig. 14a. This rule generalises `leave_secure` (Fig. 7a) allowing people to move between between

<sup>10</sup>The reasoning behind this is that you often want to tweak parameters, *e.g.* `ns`, to explore larger models and so we keep all parameters in one place: the `brs` block.

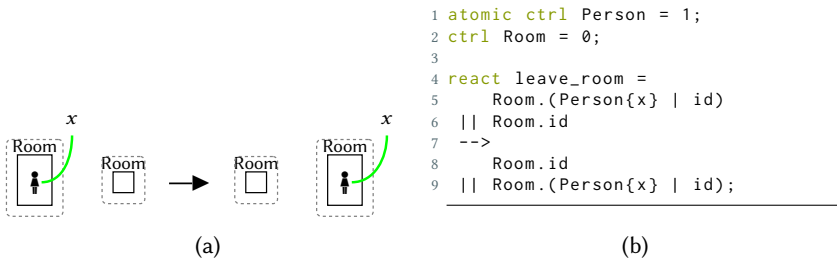
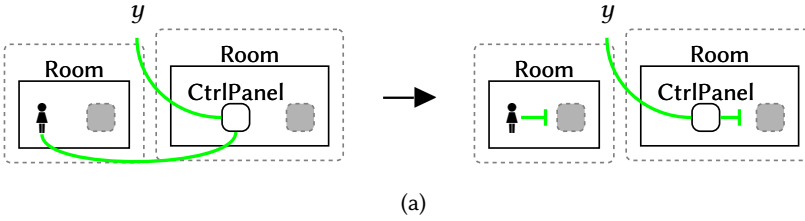


Fig. 14. Leaving a room. (a) Reaction rule `leave_room`. (b) BigraphER snippet



```

1 react fix_secure =
2   /x (Room.(Person{x} | id)
3   || Room.(CtrlPanel{x,y} | id))
4   -->
5   Room.(/x Person{x} | id)
6   || Room.(/x CtrlPanel{x,y} | id);
7
8 begin brs
9   init ...;
10  rules = [ {fix_secure}, {leave_room} ];
11 end

```

(b)

Fig. 15. Using priorities to ensure links to control panels are severed without stopping movement. (a) Rule `leave_secure`. (b) BigraphER snippet.

arbitrary rooms. Due to the connection to name `x`, this means a person connected to a `CtrlPanel` might leave the room and still be connected to the security network: an information leak. To avoid this, `leave_secure` can be applied with higher priority than `leave_room`.

A better way to model the above would be to have a single movement rule `leave_room` and a high priority rule that *fixes* the model after a movement (before any other rule applies) by severing any links to control panels in other rooms. This is shown in Fig. 15. This way we do not block other entities from moving.

BigraphER allows *rule priorities* that define a partial ordering on rule application. Each reaction rule belongs to one *priority class* (sets of rules with the same priority) and an operator `<` defines the partial order. For example, we can define  $\{X\} < \{Y\}$  meaning that rules in  $X$  are checked for matches only if no rule in  $Y$  has a match. Within a priority class rules apply non-deterministically, as before. In BigraphER syntax the classes are enumerated in the rule declaration within `{ }` brackets, e.g. in Fig. 15b there are two classes, one with higher priority containing `fix_secure` and the lower priority class containing `leave_room`.



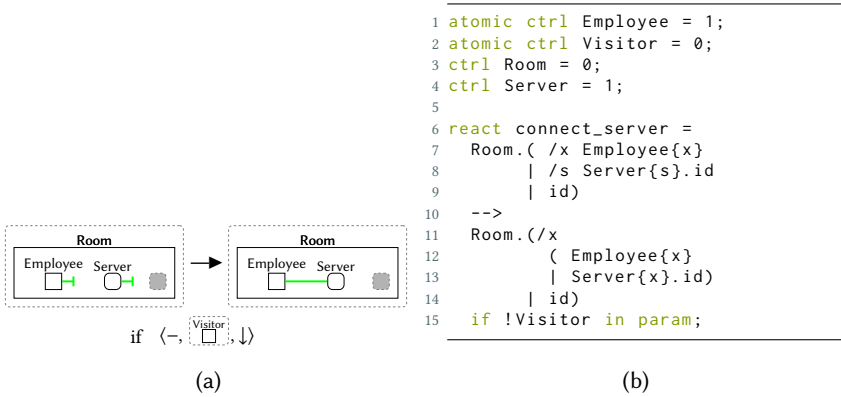


Fig. 16. Conditional rule connect\_server. (a) connect\_server rule. (b) BigraphER snippet.

**Modelling Tip 8:** Be careful when assigning rule priorities. Although priorities stop a general case applying when a specific should be applied, it also stops a general case being applied to any other matches.

While it is possible to encode priorities directly in the bigraphs through additional entities, this: i) causes larger models, ii) mixes domain specific modelling entities and control-only entities leading to more complex models, and iii) is error prone.

### 5.4 Instantaneous rules

BigraphER supports *instantaneous rules* that allow a set of reactions to occur without adding intermediate states to the transition system. The rules are specified in the standard manner and placed in an instantaneous priority class, denoted in BigraphER with ( ) instead of { }. Instantaneous priority classes are special in that they must *fully reduce* before any additional rules are called, i.e. {r<sub>1</sub>, r<sub>2</sub>} will apply either r<sub>1</sub> or r<sub>2</sub> (if possible) and then retry rules with higher priority; (r<sub>1</sub>, r<sub>2</sub>) applies *continuously* r<sub>1</sub> or r<sub>2</sub> until no further applications are possible. Instantaneous rules must be confluent, that is, they must always create the same resulting bigraph regardless of the order of application. Currently there is no tool support to check the confluence of rules, and a user must verify this themselves. The result is a more efficient transition system in which many spurious states and interleavings are removed.

### 5.5 Conditional rules


We often want to control the *contexts* under which a reaction rule can apply. This is possible in Conditional Bigraphs [3], supported by BigraphER, that allow additional conditions—specified as matches—to be attached to a rule.

A match for a rewrite has three components: the left-hand-side of the rule, a *parameter* that is everything *inside* the sites, and a *context* that is everything other than the rule and parameter. Conditional rules lets us constrain either the parameter (most commonly) or the context.

An example conditional rule is in Section 5.5.

In this example we want to avoid a leak of sensitive information by disallowing employees access to a server if there are any visitors in the room. We specify this as a condition that states the rule only applies when there are no Visitor entities in the *parameter*, i.e. the site.

We use the notation `if !X in param` to specify that  $X$  should not appear in the parameter. Similarly we can have conditionals that *require* an entity to be present, e.g. `if Y in ctx` means there must be a  $Y$  somewhere in the model that is not in the rule or parameter.

In the diagrammatic notation we show conditionals under the rules in the form `if <- , , ↓>` that denotes we do not (–) allow the (arbitrary) *bigraph*  $B$  in (any of) the sites (↓). We may also use + to enforce a bigraph must be present, and ↑ to denote context instead of the parameter.

**Modelling Tip 9:** Conditional rules allow us to restrict instantiation of sites: instead of allowing arbitrary bigraphs, conditions allow only those that do/do not match a pattern.

Due to the technical challenges detailed in [3], there are three caveats when using conditions: i) conditions cannot determine a *specific* site (or region) to match within, and need to be valid regardless of where the condition is located in the parameter, *i.e.* we cannot say site 0 does not contain a Person only no site contains a Person; ii) names in a conditional, even if they are the same as those in the match, are not guaranteed to be connected—this is due to names being structural elements (see Section 3.1) rather than global, *i.e.* we can rename in the conditional as required, and; iii) conditions cannot be nested, *i.e.* a condition cannot itself have a condition. We hope some of these restrictions can be removed in future.

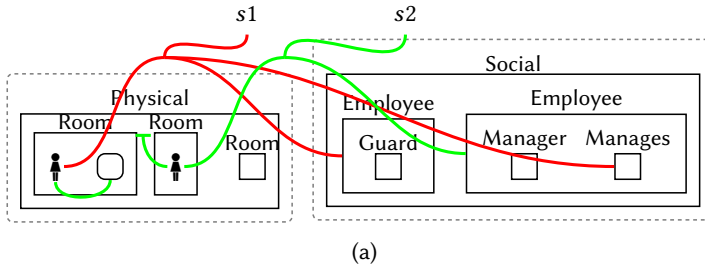
**Modelling Tip 10:** Where possible, it is better to choose a conditional rule over rule priorities. This is because conditions indicate the intent of a single rule, while priorities define relationships across the whole set of rewrite rules.

This concludes our overview of BRS, the following three sections cover a range of practical modelling advice for tackling common scenarios. Creating models is as much an art as a science, and these techniques should be seen as pieces of advice rather than absolute rules.

## 6 MULTI-PERSPECTIVE MODELLING

The use of parallel regions, with links that can cross regions, provides a way to construct models with a strong separation of concerns. We can consider each parallel region to be a *perspective*, for example we might have a region that models the *physical* characteristics of a system, while another might model *virtual* representations. Cross-region links between entities allow them to be related, *e.g.* to tie a *physical* server to *virtual* characteristics. Note that rewrite rules can be over multiple perspectives. As links are undirected hyperlinks, the relationships between perspectives form a *graph* *i.e.* we can have multiple representations of the same entity.

Multi-perspective modelling has been used to good effect in previous works. For example, the model of a cyber-physical game, Savannah [7], is split into four *design* perspectives: *physical*, that models people entering and exiting the physical game space (a playing field); *human*, that models proxemics and how human players interact to form groups; *technology*, that models how GPS technology senses and represents the physical reality, including drift and ghosting; and *computational*, that implements the rules of the game. The model was used to study behaviours observed in user trials, in particular player cognitive dissonance in certain situations. Another example is a model for a sensor system that is split into three perspectives: *physical*, where each sensor is located; *control*, information on sensor capabilities; and *application*, the devices an application requires to operate [41]. Finally, *plato-graphical* models [9] use three perspectives: *context* for a true environment; *proxy* as a representation of the context (the shadows on the wall); and *agents* that interact with the real context (environment) through the proxy.



(a)

```

1 # Perspectives
2 ctrl Physical = 0;
3 ctrl Social = 0;
4
5 # Physical
6 ctrl Room = 0;
7 atomic ctrl Server = 1;
8 atomic ctrl Person = 2;
9
10 # Social
11 ctrl Employee = 1;
12 atomic ctrl Manager = 0;
13 atomic ctrl Manages = 1;
14 atomic ctrl Guard = 0;
15
16 big multipersp =
17   Physical.(/x Room.(Person{s1,x} | Server{x}) | /y Room.Person{s2,y} | Room.1)
18   || Social.(Employee{s1}.Guard | Employee{s2}.(Manager | Manages{s1}));

```

(b)

Fig. 17. Building model with two perspectives: Physical and Social. (a) Bigraph model, coloured links distinguish hyperedges (b) BigraphER snippet.

### 6.1 Example: Multi-perspective modelling of a building

So far, our building model has been concerned mainly with the physical location of people within the building and their interactions with physical devices; we might consider this a *physical* perspective.

The people themselves have been kept abstract, *i.e.* we have a single Person entity (ignoring Adult/Child distinction), although in practice people will have different *roles* in the building, and relationships with others. We could add these directly into the existing model, perhaps by nesting specific roles, *e.g.* Manager, Guard, under a Person entity, but this mixes physical and social information. In the multi-perspective approach we can instead introduce a *social* perspective to track roles and relationships.

Figure 17 shows a simple multi-perspective model of a building extended with a social perspective. To enable cross-perspective linking, we have added an additional link to the Person entity (alternatively we could introduce the link to a nested entity: see Section 7.1). To make it clear we colour these links differently per person. For each Person in the physical perspective, they have a social representation as an Employee (assuming no visitors are allowed in this building). Not all entities will have a representation in other perspectives, *e.g.* the Servers exist physically but not socially. Within the social perspective, employees have information about roles and possible perspective-local links, *e.g.* to represent management organisation.

**Modelling Tip 11:** For multi-perspective modelling it is useful to add a top-level entity that allows a region to be referred to by name, *e.g.* Physical, Social.

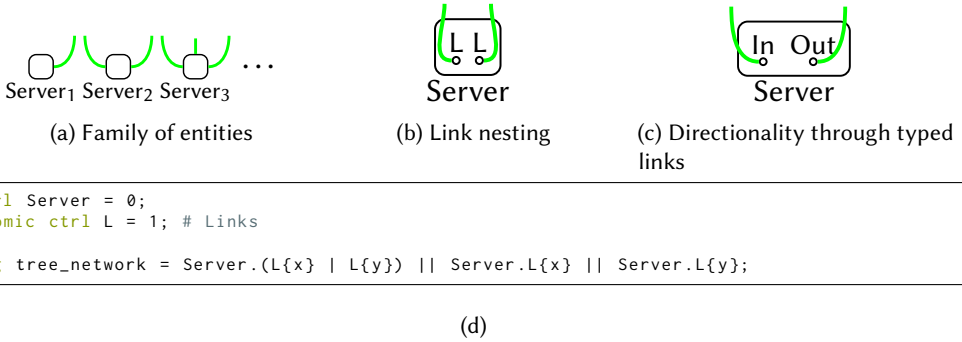


Fig. 18. Overcoming fixed arity and directed links. a) The family of Server entities with 1, 2, 3 etc. ports we want to represent. b) The new definition of Server where instead of *e.g.* 2 links, we have 2 nested L entities. c) New entities In and Out.

The multi-perspective approach is extensible. For example, an additional perspective could model an employee (virtual) staff record, *i.e.* a *Database* perspective, by linking to the Employee in the *Social* perspective. Not only does this mean the database can reflect social changes, but it can also reflect physical changes directly as all three representations share a hyperlink.

## 7 MORE ENTITIES OR MORE LINKS?

While the place and link graph are disjoint relations, the interactions between them can be used to model a wide range of scenarios. The general advice is as follows:

**Modelling Tip 12:** If it is difficult to model with places, add more link structure; if it is difficult to model with links, add more place structure.

In the next four sections we give examples of adding place structure to increase expressiveness.

### 7.1 Overcoming Fixed Arity

Recall that entities have fixed arity, *i.e.* a fixed number of ports. A common modelling paradigm is some objects have a varying number of ports, *e.g.* a server that may have  $n$  (point to point) connections.

Intuitively, we would like to model this with a *family* of Server entities, *i.e.*  $Server_n$  for  $0 \leq n \leq max\_connections$ , where each  $Server_n$  allows  $n$  links. In effect, these are sub-types, which are not supported in bigraphs. So, each of these entities would have to be specified separately, requiring an explosion in the number of rewrite rules required.

A practical way to deal with this is to introduce a new entity of arity 1 that represents a link endpoint. Now, whenever we need to add a new connection, we simply nest a new L entity and use that link to form the connection. This reduces the total number of entities required to 2 instead of a family of  $n$ , and there is now no problem to match directly on a Server entity regardless of the connections (by hiding them in a site). This approach is shown in Fig. 18a and Fig. 18b and is an example of using the *place* graph to fix what is essentially an expressiveness of *link* problem. Figure 18d shows BigraphER code using this technique to model a simple tree network topology.

## 7.2 Directed Links

It is possible to use this approach to encode directed links. For example, instead of introducing a single L entity, we introduce a pair of entities Out and In (illustrated in Fig. 18c) entities representing the source and target of a link resp.

This approach should not be confused with directed bigraphs [21], an extension to the bigraph theory where the link graph itself is directed. While they both allow direction to be specified, directed bigraphs allows *names* to also have direction<sup>11</sup>.

This is shown in Fig. 18c.

## 7.3 Ordered children

Similarly, the children of an entity are *unordered*, and merge product is commutative. If we want to order children we can again add extra entities. For example, L and R might represent the left and right argument of a subtraction function, e.g.  $\text{Sub. (L. Int}(5) \mid \text{R. Int}(2))$  meaning  $5 - 2$ .

To avoid introducing an entire family of entities when working with long lists of ordered arguments, e.g.  $\text{Arg}_1, \text{Arg}_2, \dots$ , we can encode a linked-list structure, the simplest being a single Cons entity and 1 to represent the empty cell. For example  $\text{b} \mid \text{Cons. (b' \mid \text{Cons. (b'' \mid 1))}$  is a list of length 3 (where b, b', ... are arbitrary bigraphs). Alternatively a list could be encoded through links, e.g. with links representing *pointers* to the next cell. In both cases it is possible to use rewrite rules to implement the usual functional abstractions for lists: map, fold etc.

## 8 APPLYING A RULE A FIXED NUMBER OF TIMES AND TAKING TURNS

Often we require to apply a rule, or a sequence of rules, exactly  $n$  times, which is not expressible within a standard rewriting framework. In bigraphs, we can encode rule control by *tagging*: using additional *tag* entities to mark where rules have previously been applied. Note that tagging requires rule priorities (Section 5.3) or conditional rules (Section 5.5).

We illustrate with an example. Consider the scenario where access to the vault of a building requires  $n$  people to log into the vault access system at the same time. The model is in Fig. 19. Essentially, the vault is initially Closed, then  $n$  people Login, and then the vault is Open. But, *any* number of people may be *trying* to gain access and so we have to restrict successful logins to exactly  $n$ . In other words, we cannot allow the rule for login to be applied any number of times. For simplicity we assume  $n = 2$  in this example.

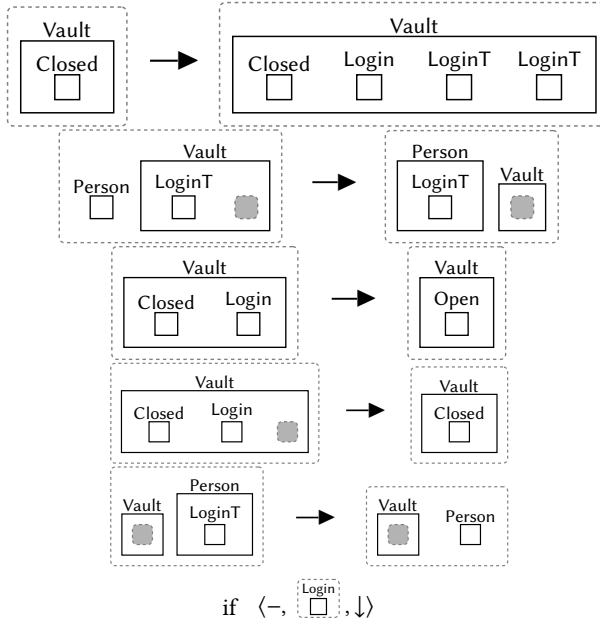
We introduce two new tags for controlling the reaction sequence: Login denotes a login sequence has started and LoginT controls the number of people required to open the vault—we use it to distinguish those that have already logged in from those that have not. The rules are modified to reflect four phases:

**Start tagging** Add a tag to denote the sequence has started and is in progress. In this case tryOpen adds an entity Login to the Vault to note the start of a login sequence. As we want a fixed number of logins, we additionally add  $n$  LoginT tokens (in this case 2, but can be changed without affecting the rest of the login mechanism).

**Apply rule(s)** Once the sequence has started, apply the rule(s)  $n$  times. In this case login allows a Person, in the same room as the vault (note use of  $\mid$  and not  $\mid\mid$ ) who has not logged in before, i.e. does not nest a LoginT tag, to perform a login by accepting the LoginT token.

**Base case(s)** The base case determines when the sequence of operations should stop, e.g. when there are no matches left. In this case open checks all LoginT tokens have been taken, allowing the vault door to open. If not enough people log in, a second base case failed stops the

<sup>11</sup>Directed bigraphs are designed to allow bisimulation congruences to be derived in the face of name aliasing rather than to solve specifically the directed link problem in models.



(a) tryOpen, login, open, failed, and clean rewrite rules.

```

1 ctrl Vault = 0;
2 ctrl Person = 0;
3
4 atomic ctrl Login = 0;
5 atomic ctrl LoginT = 0;
6 atomic ctrl Closed = 0;
7 atomic ctrl Open = 0;
8
9 # start
10 react tryOpen =
11   Vault.Closed --> Vault.(Closed | Login | LoginT | LoginT);
12
13 # apply
14 react login =
15   Person.1 | Vault.(LoginT | id) --> Person.LoginT | Vault.id;
16
17 # Base cases
18 react open =
19   Vault.(Closed | Login) --> Vault.Open;
20
21 react failed =
22   Vault.(Closed | Login | id) --> Vault.Closed @[];
23
24 # Cleanup
25 react clean =
26   Vault.id | Person.LoginT --> Vault.id | Person.1 if !(Login) in param;
27
28 begin brs
29   init ...;
30   rules = [ {clean}, { tryOpen, login, open}, {failed} ];
31 end

```

(b)

Fig. 19. Tagging example: vault login process. Two people are required to open the vault.

login sequence. Rule priorities are used to enforce failed can only be applied once we have checked all possible applications of login.

**Cleanup** The final step removes redundant tags. Here, the conditional rule `clean` removes `LoginT` entities when the `Login` sequence has ended (either successfully or with failure).

For tools that do not support conditional bigraphs, we can add an extra tag, e.g. `LoginDone`, to determine when the sequence has ended in order to control when cleanup can happen.

**Modelling Tip 13:** If you want to apply a rule sequence a fixed number of times, add tags to entities to indicate whether or not the sequence has been applied, and modify the rules so they introduce and then remove tags.

While powerful, tagging increases the number of entities and can make it less clear how entities are supposed to be used. One solution is to use tagging with instantaneous rules, see Section 5.4, this allows us to apply sequences transparently, as if in a single step. An alternative approach is to define a family of rules for each number of people we need to log in, e.g. `open_vault_1`, `open_vault_2`, ..., but as with parameterised rules, this can lead to an explosion of the total number of rules.

### 8.1 Turn taking or phases of operation

A similar, but distinct scenario is modelling *phases* of operation. For example, we might have a *movement* phase, where all people in our building can (but don't have to) move between rooms. Once *everyone* has made a movement (or idled), we might then have a *sensing* phase, where the security cameras try to detect intruders, before moving back to the movement phase.

This sort of turn-taking is well captured using multi-perspective modelling (Section 6), where a *Control* perspective can track the current state of the system, e.g. *Movement vs Sensing*. As perspectives are just additional parallel regions, it is easy to match on them, i.e. we just extend a rule  $L \rightarrow R$  to  $L \parallel C \rightarrow R \parallel C$  for some control information encoded by  $C$ .

Alternatively we can use a conditional rule (Section 5.5,  $L \rightarrow R$  if  $C$  in  $ctx$ ) If  $C$  does not change during rule execution. Keeping track of who still needs to move/sense can be done by nesting additional tokens, e.g. `Person.Move`, `Person.Sense`.

A snippet of a movement and sensing example is in Fig. 20. We show both direct matches and conditionals; in general it is good practice to use a single style only.

Key to turn taking is phase shift functions that move between movement and sensing, e.g. they model a state machine. Here we have used context conditions (Section 5.5) to check all `Person` entities have acted before swapping phase. If conditionals are not supported, then tagging (Section 8) could be used.

**Modelling Tip 14:** There may be several possible ways to model a system feature, e.g. tagging, conditional rewriting, parameterisation, etc. It is good practice to employ one consistent approach throughout the model.

## 9 FURTHER EXTENSIONS: PROBABILISTIC, STOCHASTIC, AND NON-DETERMINISTIC REWRITING

In standard BRSs, reaction rules are applied non-deterministically: *any* rule (within the current priority class) that has a match in the current state can be applied. Selection is random. Extensions to BRSs allow reaction rules to be annotated, allowing them to be applied probabilistically [4], stochastically (exiting a state at some rate) [27], or through explicit non-deterministic action choice, in the style of an Markov Decision Process [4].

---

```

1 ctrl Room = 0;
2
3 ctrl Person = 0;
4 atomic ctrl Move = 0;
5 atomic ctrl Sense = 0;
6
7 atomic ctrl Camera = 0;
8 atomic ctrl Alarm = 0;
9
10 # Phases
11 ctrl Control = 0;
12 atomic ctrl Movement = 0;
13 atomic ctrl Sensing = 0;
14
15 # Matching style
16 react move =
17   Room.(id | Person.Move) || Room.id || Control.Movement
18   -->
19   Room.id || Room.(id | Person.Sense) || Control.Movement;
20
21 # Conditional Style
22 react sense =
23   Room.(id | Camera | Person.Sense)
24   -->
25   Room.(id | Camera | Person.Move | Alarm)
26   if Control.Sensing in ctx;
27
28 react no_sense =
29   Room.(id | Person.Sense)
30   -->
31   Room.(id | Person.Move)
32   if Control.Sensing in ctx, !Camera in param;
33
34 # Phase shifts
35 react move_sense = Control.Movement --> Control.Sensing if !Person.Move in ctx;
36
37 react sense_move = Control.Sensing --> Control.Movement if !Person.Sense in ctx;

```

---

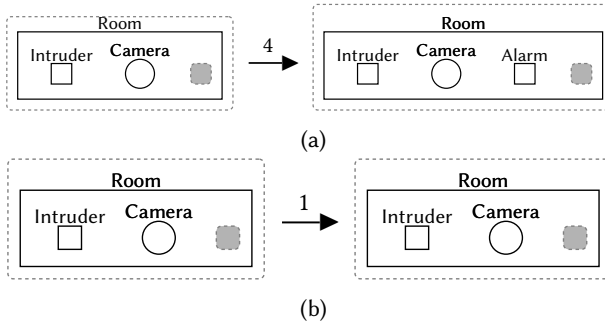
Fig. 20. Turn Taking: BigraphER snippet for movement and sensing phases.

For each extension, we annotate a rule  $r : L \rightarrow R$  with additional information, e.g.  $r_p : L \xrightarrow{2} R$  is a probabilistic rule with *weight* 2;  $r_s : L \xrightarrow{0.2} R$  is a stochastic rule with exit rate 0.2; and  $r_a : L \xrightarrow[4]{act} R$  is a rule with weight 4 if action *act* is chosen. Weights give the relative application chance for a rule. That is for rules  $r_p : L \xrightarrow{2} R$  and  $r_q : L \xrightarrow{4} R$ , if both have a match in the current state, then  $r_q$  should be twice ( $\frac{4}{2}$ ) as likely to be applied. Weightings are scaled relative to the number of matches possible, for example if  $r_p$  has two valid matches, while  $r_q$  only has one, then they are applied with the same probability.

An example probabilistic BRS is in Fig. 21. A probabilistic BRS is requested using `begin pbrs` instead of `begin brs` in the BRS definition block. In this example, we want to detect possible security threats using the security camera(s) in a room. In practice sensors, e.g. cameras, are not 100% accurate, and we want to model this uncertainty. As is common in probabilistic modelling, we use two rules representing the cases *detect* and the converse *avoid\_detect*. The weights give the relative probability of application – in this case detection is 4 times as likely as avoidance. When one Intruder is in the room *detect* applies (after normalisation) with probability 0.8. In practice, when there are many matches and priority ordering on rules etc. it becomes much harder to predict the rule probability by hand.

We give two short examples of featuring stochastic rules and non-deterministic *action* choice in Figs. 22 and 23.





```

1 atomic ctrl Intruder = 0;
2 atomic ctrl Camera = 0;
3 atomic ctrl Alarm = 0;
4 ctrl Room = 0;
5
6 react detect =
7   Room.(Intruder | Camera | id)
8   -[4]->
9   Room.(Intruder | Camera | Alarm | id);
10
11 react avoid_detect =
12   Room.(Intruder | Camera | id)
13   -[1]->
14   Room.(Intruder | Camera | id);
15
16 begin pbrs
17   init ...;
18   rules = [ {detect, avoid_detect} ];
19 end

```

(c)

Fig. 21. Probabilistic reaction rules with weights. (a) Rule detect has weight 4. (b) Rule avoid\_detect has weight 1. (c) BigraphER snippet.

Stochastic rules are like probabilistic rules but with *rates* (positive real numbers) that determine how often a particular rule should be applied. Rules with higher rates occur more often, e.g. in Fig. 22 people exit the building more often than they enter.

Non-deterministic *action* choice is offered in action bigraphs, which, like probabilistic bigraphs, specify a *weight* between the left and right of a rule, however in this case the set of rules that can be applied is affected by the action they are in. Actions are specified in the `begin abrs` block using a set syntax, e.g. `actions = [ move = {move_stay, move_room} ]` meaning that when move is chosen only reactions move\_stay and move\_room can be applied. These rules are then applied in the same style as probabilistic bigraphs.

**Modelling Tip 15:** Probabilistic, stochastic, and action bigraphs control *how* rules are applied, e.g. how often, but do not affect the rules themselves that keep the same rewriting mechanism as before.

## 10 MODEL ANALYSIS

While defining a BRS can be useful in its own right, e.g. to document design decisions, we can perform analysis through *simulation* and *model checking*. BigraphER exports a transition system (a table of transitions; possibly labelled with probabilities/rates/actions), and an initial state, that can

---

```

1 atomic ctrl Intruder = 0;
2 atomic ctrl Person = 0;
3 ctrl Room = 0;
4 ctrl Entrance = 0;
5
6 big s0 = Room.Entrance.1;
7
8 react enter = Room.Entrance.id -[0.2]-> Room.Entrance.(id | Person);
9
10 react exit = Room.Entrance.(id | Person) -[0.3]-> Room.Entrance.id;
11
12 react enter_intruder = Room.Entrance.id -[0.01]-> Room.Entrance.(id | Intruder);
13
14 begin sbrs
15   init s0;
16   rules = [ {enter, exit, enter_intruder}];
17 end

```

---

Fig. 22. Stochastic model of entrance hall: People exit the room more often than entering (rates of 0.3 vs 0.2). Intruders can enter but at a much lower rate (0.01).

---

```

1 atomic ctrl Guard = 0;
2 atomic ctrl Intruder = 0;
3 ctrl Room = 0;
4 atomic ctrl Door = 1;
5 atomic ctrl Alarm = 0;
6
7 react move_stay =
8   Room.(id | Guard) -[5]-> Room.(id | Guard);
9
10 react move_room =
11   Room.(id | Door{x} | Guard) || Room.id
12   -[1]->
13   Room.(id | Door{x}) || Room.(id | Guard);
14
15 react check_room =
16   Room.(id | Guard | Intruder) -[1]-> Room.(id | Alarm | Guard | Intruder);
17
18 react check_room_safe =
19   Room.(id | Guard) -[1]-> Room.(id | Guard) if !Intruder in param;
20
21 big s0 = /x (Room.(Door{x} | Guard) || Room.(Door{x} | Intruder));
22
23 begin abrs
24   init s0;
25   rules = [ {move_stay, move_room, check_room, check_room_safe}];
26   actions = [ move = {move_stay, move_room}, check = {check_room, check_room_safe} ];
27 end

```

---

Fig. 23. Action based (non-deterministic) model of guarded rooms. The system can decide to let guards move between rooms *or* check the room they are in. Moving between rooms is less likely than staying put (weight 5 vs weight 1). Only one of the two check rules is applicable when the action check is taken.

be analysed in tools such as PRISM [28] or STORM [25]. The initial state is required to be *ground*—a bigraph that does not contain sites or inner names. We specify the initial bigraph using `init b` (for some bigraph *b*), e.g. line 26 of Fig. 24. Before giving details of simulation and model checking, we introduce the concept of bigraph predicates, which we have found useful for analysis.

## 10.1 Bigraph Patterns

Often it is useful to label a subset of states that have a (domain specific) feature of interest, *i.e.* they satisfy a *predicate*. We specify the predicate with a *bigraph pattern* [7], which abstracts the states that may match the left-hand-side of a rewrite rule (the *pattern*). Bigraph patterns are simply (named) bigraphs that define static (state) properties. An example is in Fig. 24; the patterns are indicated by the reserved word `preds` (for *predicates*), as shown on line 28.

A state that matches pattern  $p$  is then labelled with  $p$ , and it may be labelled with more than one pattern. Pattern matching uses the same semantics as reaction rule matching, which means we have name equivalence, *i.e.*  $A\{x\}$  and  $A\{y\}$  are the same pattern.

Bigraph patterns may occur in path formulae in temporal logics, an example using the patterns from Fig. 24 is in Section 10.2.

Developing new logics and specification languages for bigraphs remains an active research area [15].

## 10.2 Model Checking

In model checking we apply *all* possible rules, in a breadth first manner, for a given state to explore all possible traces. Each new state is checked for equality with a previous state, allowing loops in the underlying transition system. BigraphER supports model checking for all extensions, *e.g.* probabilistic, stochastic, and non-deterministic rewriting by exporting the transition system in PRISM format.<sup>12</sup> To generate the transition system use the following command line:

```
bigrapher full -M <maxstates> -l <predicates.csl> -p <transition.tra> <model.big>
```

where `predicates.csl` and `transition.tra` are the exported pattern mapping and transition system resp. Once the transition system is generated, we can check temporal logics properties expressed in logics such as LTL, CTL, and PCTL [14].

**10.2.1 Example.** Servers contain sensitive data and so server rooms should be secure. A simple model is in Fig. 24.

The model has explicit Door entities that determine valid pairs of rooms. Assume that an intruder is captured by a camera—so long as they pass through a room containing at least one camera. We want to check whether there *is* an insecurity via a property such as: “is there a path to the server room that passes only through rooms that do not contain a camera?” We define three bigraph patterns: *seen* that matches a room containing at least one Camera and an Intruder; *entrance* that matches the entrance hall containing an Intruder; and *serverRoom* that matches a room containing at least a Server and an Intruder.

The transition system generated by BigraphER is in Fig. 25, and shows all possible movements between states. As we do not model capture of the intruder they are always allowed to move even if they have been seen. This means there are infinite traces possible, *e.g.*  $0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow \dots$

Although it is fairly obvious from the transition system that there is a path ( $0 \rightarrow 1 \rightarrow 3$ ) to the server room without being seen, for larger models visually inspecting the transition system is error prone and likely too complex to draw. Instead, we can express the properties logically, in a form suitable for model checking software. The property can be expressed by the (PRISM formatted) LTL formulae, using bigraph patterns, as:

```
E ["entrance" & (!"seen" U "serverRoom")]
```

<sup>12</sup>BigraphER natively supports PRISM format, but as the transition system is just a matrix it is likely to be supported by other tools with minor changes.

```

1 atomic ctrl Camera = 0;
2 atomic ctrl Server = 0;
3 atomic ctrl Intruder = 0;
4 ctrl Room = 0;
5 atomic ctrl Door = 1;
6 atomic ctrl Entrance = 0;
7
8 react move =
9   Room.(Intruder | Door{x} | id) || Room.(id | Door{x})
10  -->
11  Room.(Door{x} | id) || Room.(Intruder | id | Door{x});
12
13 # Patterns/Predicates
14 big seen = Room.(Intruder | Camera | id);
15 big entrance = Room.(Entrance | Intruder | id);
16 big serverRoom = Room.(Server | Intruder | id);
17
18 big building = /d1/d2 (
19   Room.(Entrance | Intruder | Door{d1})
20 || Room.(Camera | Door{d1} | Door{d2})
21 || Room.(Door{d1} | Door{d2})
22 || Room.(Door{d2} | Server));
23
24
25 begin brs
26   init building;
27   rules = [{move}];
28   preds = {seen, entrance, serverRoom};
29 end

```

Fig. 24. Secure building model: Can we reach a server without passing a camera?

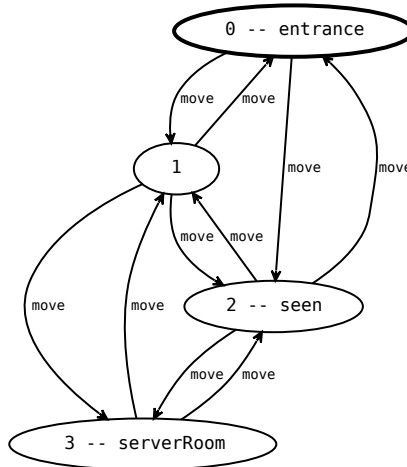


Fig. 25. Transition system generated by the model of Fig. 24. Bold state is the starting state, numbers are state numbers, and labels are predicates.

That is: there *exists* (E) a path where initially entrance holds, and for this state<sup>13</sup>, and all future states, seen does not hold *until* serverRoom becomes true. The property holds as expected because of the non-secured room (*i.e.* a room without a camera) between the entrance and the server room.

<sup>13</sup>This formula would not work if there was a camera in the entrance hall as seen would immediately be true.

### 10.3 Simulation

Simulation can be seen as a weaker form of the model checking approach where, instead of applying all rules at each state, we instead apply a single reaction rule (respecting priority and instantaneous rules), resulting in a single trace through the system. Simulation is particularly useful for models with large state spaces. BigraphER supports simulation for all extensions e.g. probabilistic and stochastic rewriting.

To run a simulation of model.big in BigraphER use the following command line:

```
bigrapher sim -S <maxstates> -l <predicates.csl> <model.big>
```

where predicates.csl is the name of the (exported) file that maps bigraph patterns to states.

As with model checking, properties of interest can be expressed using LTL formulae. Alternatively, each state can be exported with '-s .' which allows further analysis or visualisation<sup>14</sup>.

### 10.4 Common Errors

We end with a discussion of common errors you might see when executing your models with BigraphER and how to fix them.

**Init bigraph is not ground** When using models we assume the initial state is fully formed, i.e. does not contain sites<sup>15</sup>. Usually this means you have a non-atomic entity with a (implicit) site: `big s0 = A;` (which means `big s0 = A. id`). **Fix** Put an empty bigraph in place of the site, i.e. `big s0 = A. 1`.

**Invalid Reaction: Inner interfaces <1, {}> and <2, {}> do not match** This error means that the number of sites on the left hand side (1 in this case) does not match the number of sites on the right hand side (2 in this case), i.e. we haven't specified what should go into the second site on the right. **Fix** When the number of sites are unequal you *must* specify an instantiation map (Section 4.1) to determine the content of the sites after rewriting.

**Invalid Reaction: Outer interfaces <1, {x}> and <1, {}> do not match** Outer interfaces (number of regions and outer names) are not allowed to change during a rewrite. In the example above we have lost the name *x*, e.g. `A{x}. 1 --> 1`. **Fix** if an open name is no longer required you should make it idle on the right of a rule, e.g. `A{x}. 1 --> {x} | 1` is the correct rule. A similar error occurs if you drop a parallel region, requiring additional `|| 1` components to fix.

**Invalid Reaction: Instantiation map is not valid** This error means the instantiation map is badly formed. **Fix** ensure the map has an entry for each site on the right hand side and that entries point to valid sites on the left, e.g. a map `@ [5]` when the left hand side only has one site is invalid.

### 10.5 Further examples

We give an overview of five real-world systems we have modelled. For each, we indicate if (and how) we employed multi-perspective modelling, sharing, diagrammatic notation, rules, and analysis. With one exception, the shapes in the diagrammatic form are geometric (circle, rectangle, etc.). The exception is the mixed-reality game *Savannah* (Section 10.5.2) where shapes that represent entities in the game, such lions, impalas, and the human players, are hand-drawn.

The Bigraph code is available in the associated paper, or on the BigraphER website<sup>16</sup>, which contains over 25 examples.

<sup>14</sup>Exporting all states also works for full transition system analysis, but is less practical given the much larger systems

<sup>15</sup>or inner names.

<sup>16</sup><https://uog-bigraph.bitbucket.io/examples.html>

*10.5.1 802.11 CSMA/CA RTS/CTS communications protocol [10].* This well-known communications protocol applies to any network topology, including potentially overlapping wireless signals. The protocol employs four way RTS/CTS (ready to send/clear to send) handshaking. The system modelled includes arbitrary topologies and the possibility of the hidden node problem – when two transmitting stations cannot sense each other, thus causing a collision, which is resolved through exponential backoff.

*Multi-perspective modelling:* none.

*Sharing:* for wireless signal ranges.

*Diagrammatic notation:* shapes for types, coloured shading for different types or stage of data packets (e.g. queued, sent, CTS, RTS).

*Rules:* priorities to implement instantaneous rules; stochastic rules for rates of transmission and retransmission.

*Analysis:* model checking CSL (Continuous Stochastic Logic) properties that express quantitative, dynamic properties such as *what is the probability of successful transmission of a packet* and *what is the likelihood of a collision or being in an error state*.

*10.5.2 Mixed-reality multi-player game [7].* This ubiquitous system, called *Savannah* (also mentioned in Section 6), models the dynamic behaviour of players in a mixed-reality game in which the human players are instrumented and their real-world physical location affects their capabilities as players in the game. The game involves wildlife and different types of terrain (the Savannah), which is mapped on to the physical playing space (e.g. a field or football pitch). There is different wildlife in each terrain. The players hunt wildlife by forming and disbanding teams; how and when they do that depends on their proxemics: the personal (physical and social) space of each player.

*Multi-perspective modelling:* four perspectives – Human, Physical, Computational, and Technology.

*Sharing:* for overlapping auras (personal space).

*Diagrammatic notation:* shapes for types, coloured shading for animals and players at different stages in the game (e.g. idle lion, lion initiating an attack, lions and players in a group).

*Rules:* the main model employs standard reaction rules, additionally, there is an investigation of the use of weights, inferred from user trials, to represent GPS drift.

*Analysis:* simulation for replay of scenarios uncovered in user trials - scenarios in which some players experienced cognitive dissonance; manual inspection of rewrite rules for matches with bigraph patterns, which revealed missing relationships between some perspectives and a system design flaw.

*10.5.3 Network and network policy management [10].* The management of a network with an evolving topology, due to network events such as machines leaving and joining the network, is modelled. In addition, the system has dynamic access control policies that enforce or forbid certain behaviours and thus constrain network evolutions. Policies can be invoked or lifted, as the network evolves, so event streams include both network and policy events.

*Multi-perspective modelling:* none.

*Sharing:* for overlapping wireless signals.

*Diagrammatic notation:* shapes for types.

*Rules:* standard reaction rules that make use tagging to model invocation and lifting of policies.

*Analysis:* runtime monitoring of bigraph patterns that express static properties such as *the current configuration complies with the current policies*.

*10.5.4 Large-scale sensor network infrastructures [41].* A model of sensor network infrastructures that enable investigations of how requirements for a sensor network can be met, individually and collectively, and can continue to be met, in the context of large-scale, evolving network and device

configurations. The exemplar is an urban sensor network infrastructure with two applications: environmental monitoring and structural and (material) health of buildings and bridges.

*Multi-perspective modelling*: three perspectives – Physical, Data, and Services (requirements for applications).

*Sharing*: for overlapping wireless signals.

*Diagrammatic notation*: shapes for types, coloured shading to distinguish whether a node is in use or in failure, and for different types of sensor.

*Rules*: standard reaction rules; stochastic rules for failure rates and repair rates.

*Analysis*: simulations using actual data streams and events generated by the Cooja network emulator/simulator [36]; runtime monitoring of bigraph patterns that express static properties such as *there are sufficient nodes available in every network partition* and model checking LTL (Linear Time Logic) and CSL (Continuous Stochastic Logic) properties that express dynamic properties such as *pollution, temperature, and humidity data are delivered when pollution levels exceed a threshold* and *the probability of a node to fail within a certain time interval, while serving an app, is below a threshold*. This application used a custom tool, based on BigraphER's bigraph manipulation library's OCaml interface<sup>17</sup>, to allow bigraphs to update in response to simulated data streams, *i.e.* events were converted to reaction rules and applied rather than BigraphER selecting the rules (as in the standard simulation mode).

**10.5.5 CAN programming language for BDI agents [5]**. The operational semantics for CAN (Conceptual Agent Notation), an agent system programming language, is encoded in bigraphs. The structural encoding is natural, with bigraph reaction rules corresponding directly to the (semantics) inference rules. A motivation for the encoding is to be able to verify agent requirements. The exemplar is a case-study based on UAV (unmanned aerial vehicles).

*Multi-perspective modelling*: perspectives for B,D, and I – Beliefs, Desires, Intentions, and for Plans.

*Sharing*: none.

*Diagrammatic notation*: shapes for types, coloured shading for plan success or failure.

*Rules*: parameterised entities and rules, conditional rules, and rule priorities.

*Analysis*: bigraph patterns to express static properties such as *there is goal corresponding a given task* and model checking CTL properties dynamic properties such as *the goal corresponding to a given task is persistent* and *two sensing tasks can always be completed regardless of their interleaving*.

## 11 WHEN TO USE BIGRAPHS

We have shown bigraphs to be an expressive modelling approach applicable to a wide range of scenarios such as the examples in Section 10.5. While they are universal, they are not always the best tool. In this section we comment on the types of problems we think are best suited to Bigraphs, but you should feel free to experiment.

Bigraphs are based around *relationships*, both spatial and non-spatial, between entities and are well suited to model situations where relationships change over time. For example, the networking examples of Section 10.5 often allow dynamic connectivity between nodes. Bigraphs are often much less suited for models with large amounts of state, and these are often better described with tools like Event-B [1] that describe how states evolve over time.

While we support parameterised entities in BigraphER (Section 5.1) to enable some basic support for primitive data-types, *e.g.* integers, bigraphs are not well suited to data-intensive applications that require, for example, filtering of a system based on values. They are much more suited to symbolic analysis data, *e.g.* showing properties of commutativity of data-types.

<sup>17</sup><https://uog-bigraph.bitbucket.io/docs/bigraph/index.html>

Bigraphs are primarily a *system* design tool that determines how a design may respond to particular inputs/environments. Given our analysis is largely based on model checking, not proof, bigraphs are not currently a good tool for *code verification* as found in, for example, Isabelle/HOL [35], or Lean [16]. Similar system design is popular with tools such as TLA<sup>+</sup> [29], but this lacks the diagrammatic notation and is more state-based rather than relation-based. Although stylistically quite different, there are overlaps with existing graph transformation technology and bigraphs can be applied in similar areas [24].

Model checking is well known to sometimes produce a large number of states, which limits the scale of systems we can analyse. This can be mitigated in places, *e.g.* via bounded model checking [8] and executing models at runtime [12] (to only analyse the *actual* path, not all possible paths). Bigraphs are an active area of research and it is likely new analysis techniques will become available in future. For example, the theory already includes methods for computing equivalences (bisimulations) between agents (bigraphs) [30] that could be used to reduce the search space, but no implementation exists. Likewise techniques from graph rewriting such as confluence checking [38] and inductive proofs [19] would enable a much wider range of systems to be modelled.

## 12 CONCLUSIONS

Bigraphs are a versatile modelling formalism, capable of describing a wide range of systems and their dynamics. At their core is the juxtaposition of two relations: placement, described by a place graph (a forest), and connectivity, described by a hypergraph. The ability to connect entities in multiple ways not only models realistic scenarios, in that what you can do might depend both on *where* you are, and who you might be *connected* to, but leveraging both together can also overcome modelling challenges such as fixed arity constraints, ordering children, as well as enabling multi-perspective modelling.

Bigraphical reactive systems (BRSs) allow models to evolve over time. Dynamics are specified through *user specified* reaction rules that rewrite (sub-)bigraphs by bigraphs. We have shown how our extensions of priority, conditional, instantaneous, probabilistic, and stochastic rewriting, coupled with control-entities such as tagging, make BRS a powerful and expressive modelling framework.

Model analysis can be performed through simulation and model checking by generating a transition system with bigraphs as states and reaction rules as transitions.

This tutorial has foregone much of the theoretical aspects of bigraphs in favour of practical modelling techniques; key contributions are our Modelling Tips and examples expressed in BigraphER. As Milner noted:

“The model [bigraphs] is only a proposal; it can only become a foundational model for ubiquitous computing if it survives serious experimental application. For the latter, it must be seen to yield language for programming and simulation, and equipped with appropriate mechanised tools for analysis, such as model checking.” [33]

With BigraphER, the programming and tooling support is now available: it is time for the *application*. We hope this tutorial enables modellers to learn the formalism quickly, benefit from bigraphs in their work, and develop these “serious” experimental applications.

## ACKNOWLEDGEMENTS

This work is supported by the Engineering and Physical Sciences Research Council, under PE-TRAS SRF grants MAGIC and FARM (EP/S035362/1), S4: Science of Sensor Systems Software (EP/N007565/1) and an Amazon Research Award on Automated Reasoning.



## A MODELLING TIPS

- (1)  $\parallel$  and  $|$  allow us to build bigger bigraphs from smaller. Use  $\parallel$  to model distinct bigraphs and  $|$  for merging bigraphs.
- (2) Use  $1$  to indicate “no possibility of any children” and  $\text{id}$  to indicate “zero or more children”.
- (3) Use prefixes/suffixes in entity names (in textual format) and colours and shading (in diagrammatic format) to indicate relationships between states or stages of a process.
- (4) Use a named, open link to indicate “this link potentially connects elsewhere”, and a closed link to indicate “only these entities are connected”. The specific names used do not matter.
- (5) Sites are abstractions over bigraphs, i.e. they are bigraph variables that can be instantiated with a bigraph. Sites should be used when defining general rules that apply in many situations.
- (6) Careful consideration needs to be given to duplicating sites when the bigraphs being duplicated contain links: when a site is duplicated that contains links, the links remain connected. For example, if we copy  $A\{x\}$  to obtain  $A\{x\} \mid A\{x\}$ , then both  $A$  entities are connected in the result.
- (7) Parameterised rules are syntactic sugar for a set of underlying rules, so use them sparingly as each new rule increases the work needed for system analysis. In practice, this affects how you describe entities. For example, it is better to define  $\text{Camera}.\text{ID}(1)$ , which can be abstracted by a site whenever the identifier is unimportant, e.g.  $\text{Camera}.\text{id}$ , rather than  $\text{Camera}(1)$  which requires a family of rules *every* time a rule uses a  $\text{Camera}$ .
- (8) Be careful when assigning rule priorities. Although priorities stop a general case applying when a specific should be applied, it also stops a general case being applied to any other matches.
- (9) Conditional rules allow us to restrict instantiation of sites: instead of allowing arbitrary bigraphs, conditions allow only those that do/do not match a pattern.
- (10) Where possible, it is better to choose a conditional rule over rule priorities. This is because conditions indicate the intent of a single rule, while priorities define relationships across the whole set of rewrite rules.
- (11) For multi-perspective modelling, it is useful to add a top-level entity that allows a region to be referred to by name, e.g.  $\text{Physical}$ ,  $\text{Social}$ .
- (12) If it is difficult to model with places, add more link structure; if it is difficult to model with links, add more place structure.
- (13) If you want to apply a rule sequence a fixed number of times, add tags to entities to indicate whether or not the sequence has been applied, and modify the rules so they introduce and then remove tags.
- (14) There may be several possible ways to model a system feature, e.g. tagging, conditional rules, parameterisation, etc. It is good practice to employ one consistent approach throughout the model.
- (15) Probabilistic, stochastic, and action bigraphs control how rules are applied, e.g. how often, but they do not affect the rules themselves, which retain the same rewriting mechanism as before.

## B UNORDERED PORTS IN ABSTRACT BIGRAPHS

We have not presented details of the mathematical theory of bigraphs in this paper, which are available elsewhere e.g. [34]. However, one aspect of abstract bigraphs requires some discussion. As stated earlier, concrete bigraphs associate names (identifiers) with entities and closed links, whereas in abstract bigraphs, names are absent. The names are referred to as *support*. For example,

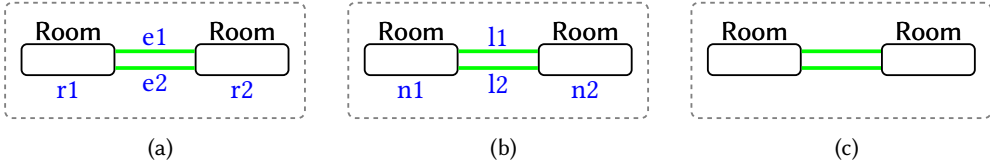


Fig. 26. (a): Concrete bigraph with two named Rooms ( $r1$  and  $r2$ ) and two links named  $e1$  and  $e2$ . (b) Concrete bigraph with two named Rooms ( $n1$  and  $n2$ ) and two links named  $l1$  and  $l2$ . (c) Abstract bigraph with no named entities or links that captures the support-equivalence class of concrete bigraphs (a) and (b).

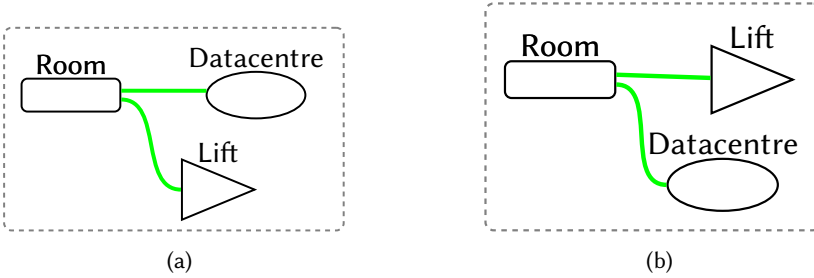


Fig. 27. (a) Abstract bigraph with three entities: Room, Datacentre, and Lift. The Room entity is linked to Datacentre and to Lift. (b) Abstract bigraph with three entities: Room, Datacentre, and Lift. The Room entity is linked to Datacentre and to Lift. (a) and (b) are the same abstract bigraph when ports are not ordered.

Fig. 26 shows two concrete bigraphs (a) and (b); abstract bigraphs capture equivalence classes of bigraphs by “forgetting” names, as illustrated by the abstract bigraph (c).

In concrete bigraphs, ports are *ordered*, *i.e.* we can refer to the  $i^{\text{th}}$  port. This may not be so apparent in the diagrammatic form, and representation of port ordering in diagrammatic form was never discussed by Milner, but we can easily define an ordering for any diagrammatic form *e.g.* scanning left to right, top to bottom. For example, in Fig. 26,  $e1$  is the first (closed) link in concrete bigraph (a) and  $l1$  is the first (closed) link in concrete bigraph (b). Note that  $e1$  and  $e2$ , and  $l1$  and  $l2$ , have the same arity and are links between two Rooms.

Abstract bigraphs capture support-equivalence classes of concrete bigraphs. In [34] Definition 2.4, this equivalence, between two concrete bigraphs, is defined by a pair of bijections  $\rho_V$  and  $\rho_E$ , on entities and edges, respectively, that “respects the structure”, *i.e.* the controls. However, these bijections induce a further bijection,  $\rho_P$ , on ports. Only one publication by Milner gives details of  $\rho_P$ , which is on page 16 of [34]. Here, the definition is given as  $\rho_P((v, i)) = (\rho_V(v), i)$ . Note the indices are *not* mapped and so port ordering is preserved. We find this an anomaly, our intuition being that in abstract bigraphs when we “forget” link names we should also “forget” link order—otherwise, we have not really forgotten names. We suggest the definition should be  $\rho_P((v, i)) = (\rho_V(v), \rho_n(i))$ , for some port order bijection  $\rho_n$ . Adopting this definition would mean the two abstract bigraphs in Fig. 27 are the same, which we propose is more appropriate for practical modelling.

Milner referred to moving between concrete and abstract bigraphs<sup>18</sup>, though no examples are given. We suggest that regardless of whether ports are ordered in abstract bigraphs, any transformation sequence concrete- $\rightarrow$ abstract- $\rightarrow$ concrete cannot be guaranteed to return to the original concrete bigraph because the names are collected as a set, not a list. Therefore, we would

<sup>18</sup>“In later chapters we shall move back and forth between concrete and abstract bigraphs, according to whether or not we need to identify support elements.” pg. 27 [34].

need an equivalence induced by a port order bijection. For example, the two concrete bigraphs in Fig. 27 are equivalent under a port order bijection.

We have been unable to find any examples in the literature that rely on port ordering in abstract bigraphs and none of the examples in this paper require port ordering. We note that several tools and encodings e.g. [23, 31, 40], assume that ports are not ordered (in abstract bigraphs). If abstract bigraphs did not have port orderings, then an ordering could always be introduced by adding the ordering information through more structure (similar to that described in Section 7.3).

## REFERENCES

- [1] Jean-Raymond Abrial. 2010. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press. <https://doi.org/10.1017/CBO9781139195881>
- [2] F. Alrimawi, L. Pasquale, and B. Nuseibeh. 2019. On the Automated Management of Security Incidents in Smart Spaces. *IEEE Access* 7 (2019), 111513–111527. <https://doi.org/10.1109/ACCESS.2019.2934221>
- [3] Blair Archibald, Muffy Calder, and Michele Sevegnani. 2020. Conditional Bigraphs. *13th International Conference on Graph Transformation, Lecture Notes in Computer Science* 12150 (2020), 3–19. [https://doi.org/10.1007/978-3-030-51372-6\\_1](https://doi.org/10.1007/978-3-030-51372-6_1)
- [4] Blair Archibald, Muffy Calder, and Michele Sevegnani. 2022. Probabilistic Bigraphs. *Formal Aspects of Computing* 34 (2022), Issue 2. <https://doi.org/10.1145/3545180>
- [5] Blair Archibald, Muffy Calder, Michele Sevegnani, and Mengwei Xu. 2022. Modelling and verifying BDI agents with bigraphs. *Sci. Comput. Program.* 215 (2022), 102760. <https://doi.org/10.1016/j.scico.2021.102760>
- [6] Blair Archibald, Michele Sevegnani, and Muffy Calder. 2022. Practical Modelling with Bigraphs: Model Files. <https://doi.org/10.5281/zenodo.7188769>
- [7] Steve Benford, Muffy Calder, Tom Rodden, and Michele Sevegnani. 2016. On Lions, Impala, and Bigraphs: Modelling Interactions in Physical/Virtual Spaces. *ACM Trans. Comput. Hum. Interact.* 23, 2 (2016), 9:1–9:56. <https://doi.org/10.1145/2882784>
- [8] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 1999. Symbolic Model Checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1579)*, Rance Cleaveland (Ed.). Springer, 193–207. [https://doi.org/10.1007/3-540-49059-0\\_14](https://doi.org/10.1007/3-540-49059-0_14)
- [9] Lars Birkedal, Søren Debois, Ebbe Elsberg, Thomas T. Hildebrandt, and Henning Niss. 2006. Bigraphical Models of Context-Aware Systems. *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS, Lecture Notes in Computer Science* 3921 (2006). [https://doi.org/10.1007/11690634\\_13](https://doi.org/10.1007/11690634_13)
- [10] Muffy Calder, Alexandros Kolioussis, Michele Sevegnani, and Joseph S. Svntek. 2014. Real-time verification of wireless home networks using bigraphs with sharing. *Science of Computer Programming* 80 (2014), 288–310. <https://doi.org/10.1016/j.scico.2013.08.004>
- [11] Muffy Calder and Michele Sevegnani. 2014. Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing. *Formal Aspects of Computing* 26, 3 (2014), 537–561. <https://doi.org/10.1007/s00165-012-0270-3>
- [12] Radu Calinescu and Shinji Kikuchi. 2010. Formal Methods @ Runtime. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems - 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6662)*, Radu Calinescu and Ethan K. Jackson (Eds.). Springer, 122–135. [https://doi.org/10.1007/978-3-642-21292-5\\_7](https://doi.org/10.1007/978-3-642-21292-5_7)
- [13] Alessio Chiapperrini, Marino Miculan, and Marco Peressotti. 2020. Computing Embeddings of Directed Bigraphs. In *Graph Transformation - 13th International Conference, ICGT 2020 (Lecture Notes in Computer Science, Vol. 12150)*, Fabio Gadducci and Timo Kehrer (Eds.). Springer, 38–56. [https://doi.org/10.1007/978-3-030-51372-6\\_3](https://doi.org/10.1007/978-3-030-51372-6_3)
- [14] Edmund M. Clarke and E. Allen Emerson. 1981. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981 (Lecture Notes in Computer Science, Vol. 131)*, Dexter Kozen (Ed.). Springer, 52–71. <https://doi.org/10.1007/BFb0025774>
- [15] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. 2005. Spatial Logics for Bigraphs. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3580)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, 766–778. [https://doi.org/10.1007/11523468\\_62](https://doi.org/10.1007/11523468_62)
- [16] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9195)*,

- Amy P. Felty and Aart Middeldorp (Eds.). Springer, 378–388. [https://doi.org/10.1007/978-3-319-21401-6\\_26](https://doi.org/10.1007/978-3-319-21401-6_26)
- [17] Søren Debois and Troels Christoffer Damgaard. 2005. *Bigraphs by Example*. Technical Report. IT University of Copenhagen. <https://en.itu.dk/research/technical-reports/technical-reports-archive/2005/tr-2005-61>
- [18] Jörg Desel and Gabriel Juhás. 2001. *What Is a Petri Net? Informal Answers for the Informed Reader*. Springer, 1–25. [https://doi.org/10.1007/3-540-45541-8\\_1](https://doi.org/10.1007/3-540-45541-8_1)
- [19] Johannes Dyck and Holger Giese. 2015. Inductive Invariant Checking with Partial Negative Application Conditions. In *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9151)*, Francesco Parisi-Presicce and Bernhard Westfechtel (Eds.). Springer, 237–253. [https://doi.org/10.1007/978-3-319-21145-9\\_15](https://doi.org/10.1007/978-3-319-21145-9_15)
- [20] Alexander John Faithfull, Gian Perrone, and Thomas T. Hildebrandt. 2013. Big Red: A Development Environment for Bigraphs. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 61 (2013). <https://doi.org/10.14279/tuj.eceasst.61.835>
- [21] Davide Grohmann and Marino Miculan. 2007. Directed Bigraphs. *Electr. Notes Theor. Comput. Sci.* 173 (2007), 121–137. <https://doi.org/10.1016/j.entcs.2007.02.031>
- [22] Dominik Grzelak. 2023. Bigraph Toolkit suite. *Technische Universität Dresden* (2023). <https://bigraphs.org>
- [23] Dominik Grzelak and Uwe Afmann. 2021. A Canonical String Encoding for Pure Bigraphs. *Springer Nature Computer Science* 2 (2021). <https://doi.org/10.1007/s42979-021-00552-5>
- [24] Reiko Heckel and Gabriele Taentzer. 2020. *Graph Transformation for Software Engineers - With Applications to Model-Based Development and Domain-Specific Language Engineering*. Springer. <https://doi.org/10.1007/978-3-030-43916-3>
- [25] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2022. The probabilistic model checker Storm. *Int. J. Softw. Tools Technol. Transf.* 24, 4 (2022), 589–610. <https://doi.org/10.1007/s10009-021-00633-z>
- [26] Espen Højsgaard and Arne John Glenstrup. 2011. The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems. *IT University of Copenhagen technical report* (2011). <https://en.itu.dk/research/technical-reports/technical-reports-archive/2011/tr-2011-145>
- [27] Jean Krivine, Robin Milner, and Angelo Troina. 2008. Stochastic Bigraphs. *Electr. Notes Theor. Comput. Sci.* 218 (2008), 73–96. <https://doi.org/10.1016/j.entcs.2008.10.006>
- [28] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. *Computer Aided Verification - 23rd International Conference, CAV, Lecture Notes in Computer Science* 6806 (2011), 585–591. [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
- [29] Leslie Lamport. 2002. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley. <http://research.microsoft.com/users/lamport/tla/book.html>
- [30] James J. Leifer and Robin Milner. 2000. Deriving Bisimulation Congruences for Reactive Systems. In *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1877)*, Catuscia Palamidessi (Ed.). Springer, 243–258. [https://doi.org/10.1007/3-540-44618-4\\_19](https://doi.org/10.1007/3-540-44618-4_19)
- [31] Cecile Marcon, Cyril Allignol, Celia Picard, Blair Archibald, Michele Sevegnani, , and Xavier Thirioux. 2024?. Modelling Bigraphs with Coq. (2024?).
- [32] Robin Milner. 2008. Bigraphs and Their Algebra. *Electron. Notes Theor. Comput. Sci.* 209 (2008), 5–19. <https://doi.org/10.1016/j.entcs.2008.04.002>
- [33] Robin Milner. 2008. Lecture notes on Bigraphs: a Model for Mobile Agents. <https://www.cl.cam.ac.uk/archive/rm135/Bigraphs-Notes.pdf>
- [34] Robin Milner. 2009. *The Space and Motion of Communicating Agents*. Cambridge University Press.
- [35] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283. Springer. <https://doi.org/10.1007/3-540-45949-9>
- [36] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. 2006. Cross-Level Sensor Network Simulation with Cooja. In *Proceedings. 2006 31st IEEE conference on local computer networks*. IEEE, 641–648.
- [37] Gian Perrone, Søren Debois, and Thomas T. Hildebrandt. 2012. A model checker for Bigraphs. In *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012*, Sascha Ossowski and Paola Lecca (Eds.). ACM, 1320–1325. <https://doi.org/10.1145/2245276.2231985>
- [38] Detlef Plump. 1993. *Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence*. Wiley-Blackwell, United States, 201–213.
- [39] Michele Sevegnani and Muffy Calder. 2015. Bigraphs with sharing. *Theor. Comput. Sci.* 577 (2015), 43–73. <https://doi.org/10.1016/j.tcs.2015.02.011>
- [40] Michele Sevegnani and Muffy Calder. 2016. BigraphER: Rewriting and Analysis Engine for Bigraphs. *Computer Aided Verification - 28th International Conference, CAV 2016* 9780 (2016), 494–501. [https://doi.org/10.1007/978-3-319-41540-6\\_27](https://doi.org/10.1007/978-3-319-41540-6_27)

- [41] Michele Sevegnani, Milan Kabác, Muffy Calder, and Julie A. McCann. 2018. Modelling and Verification of Large-Scale Sensor Network Infrastructures. In *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*. IEEE Computer Society, 71–81. <https://doi.org/10.1109/ICECCS2018.2018.00016>
- [42] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. 2018. On the Interplay Between Cyber and Physical Spaces for Adaptive Security. *IEEE Transactions on Dependable and Secure Computing* 15, 3 (2018), 466–480. <https://doi.org/10.1109/TDSC.2016.2599880>
- [43] Lisa Walton and Michael F. Worboys. 2012. A Qualitative Bigraph Model for Indoor Space. In *Geographic Information Science - 7th International Conference, GIScience 2012*. 226–240. [https://doi.org/10.1007/978-3-642-33024-7\\_17](https://doi.org/10.1007/978-3-642-33024-7_17)