# BigraphER: rewriting and analysis engine for bigraphs

Michele Sevegnani and Muffy Calder

School of Computing Science, University of Glasgow

**Abstract.** BigraphER is a suite of open-source tools providing an efficient implementation of rewriting, simulation, and visualisation for bigraphs, a universal formalism for modelling interacting systems that evolve in time and space and first introduced by Milner. BigraphER consists of an OCaml library that provides programming interfaces for the manipulation of bigraphs, their constituents and reaction rules, and a command-line tool capable of simulating Bigraphical Reactive Systems (BRSs) and computing their transition systems. Other features are native support for both bigraphs and bigraphs with sharing, stochastic reaction rules, rule priorities, instantiation maps, parameterised controls, predicate checking, graphical output and integration with the probabilistic model checker PRISM.

## 1 Introduction

Bigraphs were first introduced by Robin Milner as a universal mathematical model for representing the spatial configuration of physical or virtual objects, their interaction capabilities and temporal evolution. They were subsequently extended to stochastic bigraphs [11] and bigraphs with sharing [16], and have been applied in areas such as wireless protocols, home network management, mixed reality systems, cloud computing, security and as meta-models to encode process calculi (*e.g.* Mobile Ambients, CSS).

BigraphER is a modelling and reasoning environment for bigraphs consisting of an OCaml library and a command-line tool. The functionality includes:

- native support for both bigraphs and bigraphs with sharing;
- a rewrite engine with support for stochastic reaction rules, rules with instantiation maps, rule priorities, (stochastic) simulation and exhaustive state space exploration;
- predicate checking;
- efficient matching engine based on SAT (used to implement rewriting and predicate checking);
- support for parameterised controls and parameterised reaction rules;
- export labelled transition systems to probabilistic model checker PRISM [12];
- graphical output of bigraphs, reaction rules and transition systems (see Fig. 1 (right) for an example bigraph and graphical layout).

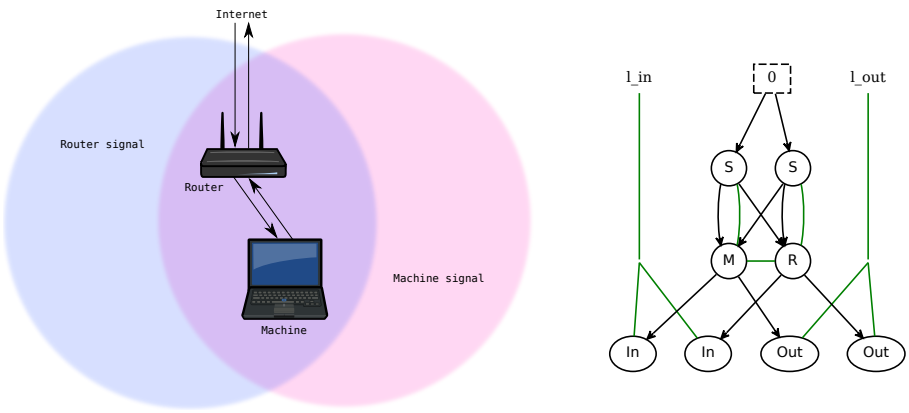


**Fig. 1.** Left: wireless network with a router and a machine; signal coverage is represented by coloured circles. Right: corresponding bigraphical representation automatically generated by BigraphER (S = signal, R = router, M = machine).

**Example Applications.** While many early applications of bigraphs have been to meta-modelling, *e.g.* for encodings of the $\pi$-calculus, $\lambda$-calculus, and CCS (Calculus of Communicating Systems), applications in other domains are recently beginning to emerge. Some examples are: security for cyber-physical systems [18], quantitative analysis of biological processes [11], cloud computing [19], and a framework to control systems of networked mobile robotic systems [14]. BigraphER has been used to specify and analyse a wide range of case studies in many different application domains: wireless network protocols [6], wireless mesh networks [4], run-time policy management for domestic networks [5], and human-computer interaction in mixed-reality systems [2]. Example analysis has ranged from detecting basic "programming" errors (*e.g.* through type checking) in [2], to generation of example state spaces, and run-time checking of invariants (*i.e.* predicates), implemented on a router in [5].

**Related Tools.** BigMC [15] is an explicit-state model checking tool for BRSs based on the BPL matching engine [3]. Currently, it does not support stochastic bigraphs nor bigraphs with sharing and can only check reachability properties. Big Red [9] is a visual editor for bigraphs and bigraphical reaction rules implemented as an Eclipse plugin; it does not implement rewriting. DBtk [1] is an implementation of matching for directed bigraphs, a variant of bigraphs with a directed link structure; there is no support for rewriting and BRS execution.

## 2 Bigraphical Reactive Systems – Overview

A *bigraph* [13] is a pair of relations over the same set of *nodes*: a directed forest, called *place graph*, representing topological space in terms of node containment and a hypergraph, called *link graph*, representing the interactions and (non-spatial) relationships among nodes. There is both an algebraic and graphical

form. The graphical representation of an example bigraph is in Fig. 1 (right); it models the simple network in Fig. 1 (left) with a router, a machine, and the range of their wireless signals.

Nodes are indicated by circles and ovals and are assigned a type called *control* indicated here by S (for signals), M (the machine), R (the router), *etc.* The place graph is specified by black arrows. *Bigraphs with sharing* [16] extend the original theory by defining the place graph as a Directed Acyclic Graph (DAG), thus allowing a natural representation of overlapping or intersecting locations. For instance, the M-node in the example is contained by *both* nodes of control S, meaning the machine is in a spatial location covered by both wireless signals. The link graph is represented by green edges called *links*. Links may be only partially specified, in which case they connect a *name*. Names are links (or potential links) to other bigraphs representing the external environment or context. By convention, names are drawn above the bigraph. In the example, names *l_in* and *l_out* are used to name incoming and outgoing (potential) links to remote resources. The number of links of a node, also called *arity*, depends on its control, *i.e.* entities with the same control have the same number of links. Dashed rectangles denote *regions* of adjacent parts of the system and *sites* are used to model parts of the model that have been abstracted away (see Fig. 3 (top)). A bigraph with node identifiers is said to be *concrete*. When all the identifiers are ignored, we obtain an *abstract* bigraph which can be interpreted as an equivalence class of bigraphs with the same structure.

A BRS consists of a set of *reaction rules* together with an initial bigraph on which the rules operate. In *stochastic bigraphs* [11], a rate is associated with each rule.

## 3 BigraphER Specification Language

The BigraphER specification language almost corresponds to the standard algebraic notation for bigraphical expressions [13,16]. In the following, we highlight some of the distinctive features of the BigraphER language by presenting a simple model for wireless networks inspired by [5]. The model is specified by the code in Fig. 2. A valid BRS model consists of four separate blocks of definitions: a *signature* containing all the controls in the model, a set of bigraphs, a set of reaction rules and a *reactive system* specifying the initial state of the BRS, the priority hierarchy among reaction rules and a set of predicates.

Controls are defined in lines 1-2 by using keyword `ctrl`. The integer on the right-hand side of each definition indicates the arity of each control. The keyword `atomic` specifies that a node may not contain other nodes. Bigraph definitions are in lines 4-8. Line 5 defines bigraph `s0`. Expression `M{w,s}` denotes a node of control `M` with names `w` and `s`. Operators `.` and `|` denote nesting and merge product, respectively. Nesting is the operation allowing to place a bigraph inside another one; merge product is the operation placing two bigraphs side-by-side inside the same region. Closures like `/s0` indicate that a link has no names (see link between `M` and `S`). Sharing is introduced by ternary operator `share ... by ... in ...`. The first argument specifies the entities to be

```
1  ctrl M = 2; ctrl R = 2; ctrl S = 1;
2  atomic ctrl In = 1; atomic ctrl Out = 1; atomic ctrl Block = 1;
3
4  big links = In{l_in} | Out{l_out};
5  big s0 = /s0 /s1 (share (/w (M{w,s0}.links || R{w,s1}.links))
6                   by ([{0,1}, {0,1}], 2)
7                   in (id{s0,s1,l_in,l_out} | S{s0} | S{s1}));
8  big is_in_blocked = M{w,s}.(In{l} | Block{l} | id);
9
10 react block_in =
11   M{w,s}.(In{l} | id) --> M{w,s}.(In{l} | Block{l} | id);
12
13 react leave_net =
14   /s (share (M{w,s} || id) by ([{0, 1}, {1}], 2) in (id(1,{w,s}) || S{s}))
15   -->  ({w} || 1 || 1 || 0 || 0);
16
17 brs
18   init s0;
19   rules = [ { block_in, leave_net } ];
20   preds = { is_in_blocked };
21 endbrs
```

**Fig. 2.** Specification of a BRS in the BigraphER language.

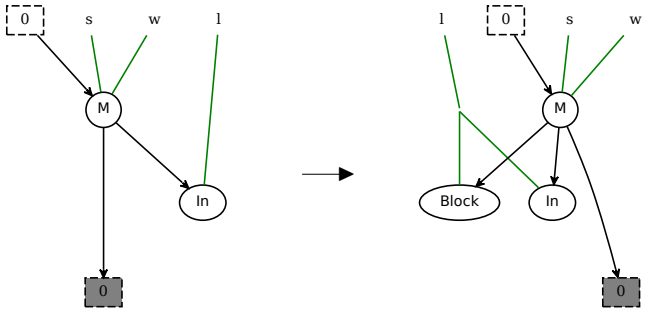


**Fig. 3.** Reaction rule `block_in` for blocking a machine's incoming traffic.

shared, *e.g.* machine `M` and router `R`. The second argument specifies how they are shared: `{0, 1}` indicates that `M` is shared by the first and the second signals (counting from left to right). The third argument specifies the entities containing the shared entities, *e.g.* signals `Ss0` and `Ss1`. The graphical representation of `s0`, automatically generated by BigraphER is shown in Fig. 1 (right).

The code in lines 10-11 defines reaction rule `block_in`. Operator `-->` is used to separate the left-hand side from the right-hand side of the rule. Expression `id` indicates the identity bigraph, *i.e.* the bigraph with one site inside one region. This reaction rule models a firewall rule blocking a machine's incoming traffic. The corresponding graphical representation is in Fig. 3. Reaction rule `leave_net` defined in lines 13-15 models a machine leaving the network.

Finally, lines 17-21 contain the reactive system definition. A BRS is defined by construct `brs ... endbrs`. Keyword `init` specifies the initial state of the system. In the example, this is bigraph `s0`. Construct `rules = [...]` defines a

list of priority classes in descending order of priority. A priority class is specified by construct `{...}` and may only contain reaction rules identifiers. Construct `preds = {...}` defines a set of predicates. Predicate `is_in_blocked` (defined in line 8) can be used to tag states in which there are machines with blocked incoming traffic. In a more extensive model like in [5], this predicate can be used to verify network invariants after network policies are enforced by users.

This simple example highlights the main features; more complex examples including stochastic reaction rules, reducible priority classes and instantiation maps can be accessed at `http://www.dcs.gla.ac.uk/~michele/bigrapher.html`.

## 4 Components and Features

The BigraphER command-line tool is composed of three distinct modules: the compiler, the matching engine and the rewriting engine. All are coded in OCaml.

**Compiler.** The compiler translates an input source file in the BigraphER language into a run-time representation of the model. Each declaration specifies the binding of an identifier to a data type representing either a control, a bigraph or a (stochastic) reaction rule. Each bigraph is stored in memory as a pair of specialised data structures: a sparse boolean matrix encoding the DAG's adjacency matrix of the place graph, and a set of hyperedges (*i.e.* multisets with nodes and names as elements) for the representation of the link graph. Although the BigraphER language only defines abstract bigraphs, the compiler operates on the corresponding concrete bigraphs by assigning arbitrary node labellings. This is required to allow the enumeration of all distinct occurrences of a reaction rule and thus to compute *exit rates* in stochastic BRSs. Additional features are: type-checking of parameterised definitions, combinatorial generation of parametric reaction rules, graphical representation of all the bigraph defined in the input model (useful for debugging).

**Matching Engine.** The bigraph matching problem determines whether a bigraph, called *pattern*, occurs in another bigraph, called *target*. The BigraphER matching engine implements the algorithm introduced in [16]: a SAT encoding of a specialisation of the sub-graph isomorphism problem. For each instance of the problem, the matching engine generates a set of *constraints* (formulas in Conjunctive Normal Form (CNF)) encoding the instance. Solutions are then obtained by passing all the constraints through the OCaml bindings for the MiniSat solver [7]. Solutions are expressed as total maps from the nodes of the pattern to sub-sets of the nodes of target. Because the matching problem is **NP**-complete, two techniques to optimise performance have been adopted in the implementation. The first is to reduce the size of the SAT instances by applying Tseitin transformation [17] to constraints. The second is to minimise instances by exploiting the symmetries in the structure of the pattern: when enumerating all the occurrences, the automorphisms of the pattern are used to generate all the symmetric solutions starting from a computed solution. The matching engine also implements specialised constraints to support bigraph equality and predicate checking.

**Rewrite Engine.** This component computes the dynamic evolution of the (stochastic) BRS specified in the input file by iteratively applying all the reaction rules to each bigraph (state) until either a fixpoint or a user-defined bound on the number of states is reached.[1] The transition system generated by a BRS is represented internally by BigraphER as a directed graph; the Continuous Time Markov Chain resulting from a stochastic BRS as a labelled directed graph. Rule application consists of two steps: first the matching engine is queried for occurrences of the left-hand side of a reaction rule, then, for each distinct occurrence, a new state is computed by replacing the occurrence with the right-hand side of the rule (see Fig. 3). BigraphER also supports reaction rules with *instantiation maps*[2] allowing to easily duplicate or discard parts of a bigraph when a reaction rule is applied. The rewriting engine incrementally builds the state space in a breadth-first search (BFS). Support for simulation is obtained by computing only one random path of the transition system. Simulation for stochastic BRSs implements Gillespie's Stochastic Simulation Algorithm (SSA) [10]. Besides standard rule priorities, BigraphER admits *reducible classes*[3] *i.e.* priority classes in which rules are treated like rewriting within an equivalence class. This means that after applying all possible rules in an arbitrary order only a canonical form is stored. This feature allows, for instance, to reduce the number of intermediate states generated by the application of instantaneous stochastic reaction rules. Predicates expressed as matches are checked during the generation of the transition system: every time a new state is discovered, all the predicates specified in the input model are checked against it and the labelling function is updated. The rewriting engine can return either a textual or a graphical representation of the (labelled) transition system and its states. Graphical output is computed by the open-source graph layout generator Graphviz [8]. Textual output is compatible with the PRISM probabilistic model checker, thus enabling quantitative verification for BRSs.

**OCaml Library.** This component provides programming interfaces for the data structures used internally by the BigraphER command-line tool. For instance, it allows manipulation of bigraphs and their constituents by providing implementation for the following operations: composition, tensor product, parallel product, merge product and nesting. The library also provides APIs to check predicates, construct reaction rules and apply them to rewrite bigraphs. The full library documentation can be accessed at `http://www.dcs.gla.ac.uk/~michele/docs/bigraph/index.html`.

**Technical Details and Availability.** BigraphER is free and open source (BSD) and runs on all major operating systems. It is available for download from `http://www.dcs.gla.ac.uk/~michele/bigrapher.html`.

---

[1] Note that a model may have an infinite state space.

[2] An instantiation map is a function associating each site in the right-hand side to sites in the left-hand side.

[3] The reaction rules belonging to a reducible class are assumed confluent *i.e.* they yield the same result regardless of the order in which they are applied.

**Acknowledgments.** This work was supported by EPSRC project Homework (EP/F064225/1) and an EPSRC Doctoral Prize Research Fellowship.

## References

1. Bacci, G., Grohmann, D., Miculan, M.: DBtk: A toolkit for directed bigraphs. LNCS 5728, 413–422 (2009)
2. Benford, S., Rodden, T., Calder, M., Sevegnani, M.: On lions, impala, and bigraphs: modelling interactions in physical/virtual spaces. ACM Transactions on Computer-Human Interaction (2016), in press
3. Birkedal, L., Damgaard, T.C., Glenstrup, A.J., Milner, R.: Matching of bigraphs. ENTCS 175(4), 3 – 19 (2007)
4. Boucebsi, R., Belala, F.: Towards a channels allocation scheme model for WMNs based on SBRS with sharing. In: Proceedings of MeMo 2015. p. 5 (2015)
5. Calder, M., Koliousis, A., Sevegnani, M., Sventek, J.: Real-time verification of wireless home networks using bigraphs with sharing. Science of Computer Programming 80, Part B, 288–310 (2014)
6. Calder, M., Sevegnani, M.: Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing. Formal Aspects of Computer Science 26, 537–561 (2014)
7. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and applications of satisfiability testing. pp. 502–518. Springer (2004)
8. Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz– open source graph drawing tools. LNCS 2265, 483–484 (2002)
9. Faithfull, A., Perrone, G., Hildebrandt, T.T.: Big Red: A development environment for bigraphs. In: Proceedings of GCM 2012. vol. 61 (2013)
10. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81(25), 2340–2361 (1977)
11. Krivine, J., Milner, R., Troina, A.: Stochastic bigraphs. ENTCS 218, 73–96 (2008)
12. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. LNCS 6806, 585–591 (2011)
13. Milner, R.: The space and motion of communicating agents. Cambridge University Press (2009)
14. Pereira, E., Kirsch, C., Sengupta, R., Sousa, J.: BigActors - a model for structure-aware computation. In: 4th International Conference on Cyber-Physical Systems. pp. 199–208. ACM/IEEE (2013)
15. Perrone, G., Debois, S., Hildebrandt, T.T.: A model checker for bigraphs. In: Proceedings of SAC '12. pp. 1320–1325. ACM (2012)
16. Sevegnani, M., Calder, M.: Bigraphs with sharing. Theoretical Computer Science 577, 43 – 73 (2015)
17. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic 2(115-125), 10–13 (1968)
18. Tsigkanos, C., Pasquale, L., Ghezzi, C., Nuseibeh, B.: Ariadne: Topology aware adaptive security for cyber-physical systems. In: ICSE 2015. vol. 2, pp. 729–732 (2015)
19. Yu, L., Tsai, W.T., Wei, X., Gao, J., Hildebrandt, T., Guo, X.Q.: Modeling and analysis of mobile cloud computing based on bigraph theory. In: MobileCloud 2014. pp. 67–76 (2014)