Nondeterministic Bigraphical Reactive Systems for Markov Decision Processes[⋆]

Paulius Dilkas

University of Glasgow, Glasgow, UK

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: Bigraphs · Bigraphical Reactive Systems · Probabilistic semantics · Markov decision process.

1 Introduction

1.1 Markov Decision Processes

Definition 1 (Markov decision process). *For any finite set X, let $Dist(X)$ denote the set of discrete probability distributions over X. A* Markov Decision Process *is a tuple $(S, \overline{s}, A, P, L)$, where: S is a finite set of states and $\overline{s} \in S$ is the initial state; A is a finite set of* actions*; $P : S \times A \to Dist(S)$ is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S; $L : S \to 2^P$ is a labelling with atomic propositions.*

Definition 2 (rewards). *A* reward structure *for an MDP $(S, \overline{s}, A, P, L)$ is a pair (ρ, ι), where $\rho : S \to \mathbb{R}$ is the* state reward function*, and $\iota : S \times A \to \mathbb{R}$ is the* transition reward function*.*

1.2 Bigraphs

From [3]. From PhD thesis [2]

Consider the graphical representation of a bigraph in Fig. 1. The white ellipses are the *nodes*. Each node has a type, called *control*, denoted here by the labels A, B, and C. The two white dashed rectangles represent *regions*, i.e. parents of any otherwise parentless nodes. Grey dashed rectangles are called *sites* and encode parts of the model that have been abstracted away. Nodes, sites, and regions are also known as *places*. Two places with the same parent, or two points with the same link are called *siblings*.

Definition 3 (concrete place graph with sharing). *A* concrete place graph with sharing

$$F = (V_F, ctrl_F, prnt_F) : m \to n$$

[⋆] Supported by organization x.

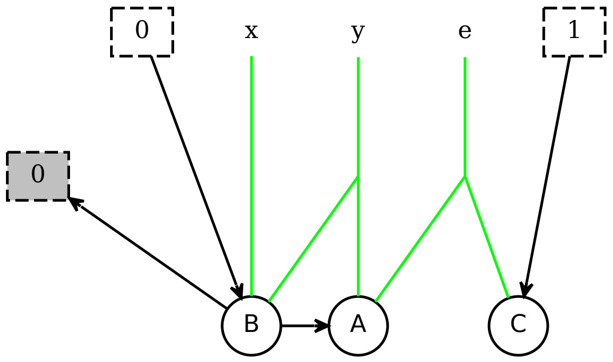


Fig. 1. An example bigraph.

is a triple having an inner interface m and an outer interface n. These index the sites and regions of the place graph, respectively. F has a finite set $V_F \subset \mathcal{V}$ of nodes, a control map $ctrl_F : V_F \to \mathcal{K}$, and a parent relation

$$prnt_F \subseteq (m \uplus V_F) \times (V_F \uplus n)$$

that is acyclic, i.e., $(v, v) \notin prnt_F^+$ for any $v \in V_F$.

Definition 4 (composition for place graphs with sharing). *If $F : k \to m$ and $G : m \to n$ are two concrete place graphs with sharing with $V_F \cap V_G = \emptyset$, their composite*

$$G \circ F = (V, ctrl, prnt) : k \to n$$

has nodes $V = V_F \uplus V_G$ and control map $ctrl = ctrl_F \uplus ctrl_G$. Its parent relation $prnt \subseteq (k \uplus V) \times (V \uplus n)$ is given by:

$$prnt \coloneqq prnt_G^{\lhd} \uplus prnt_\circ \uplus prnt_F^{\rhd},$$

where

$$\begin{aligned} prnt_F^{\rhd} &= prnt_F \rhd V_F, \\ prnt_G^{\lhd} &= V_G \lhd prnt_G, \\ prnt_\circ &= (m \lhd prnt_G) \circ (prnt_F \rhd m). \end{aligned}$$

Definition 5 (tensor product for place graphs). *If $G_0 : m_0 \to n_0$ and $G_1 : m_1 \to n_1$ are two concrete place graphs with sharing with $V_F \cap V_G = \emptyset$, their tensor product*

$$G_0 \otimes G_1 = (V, ctrl, prnt) : m_0 + m_1 \to n_0 + n_1$$

has nodes $V = V_{G_0} \uplus V_{G_1}$ and control map $ctrl \coloneqq ctrl_{G_0} \uplus ctrl_{G_1}$. Its parent relation $prnt \subseteq [(m_0 + m_1) \uplus V] \times [V \uplus (n_0 + n_1)]$ is defined as

$$prnt_{G_0} \uplus prnt_{G_1}^{(m_0,n_0)},$$

where

$$\begin{aligned}
prnt_{G_1}^{(m_0,n_0)} = & \{(v,w) \mid (v,w) \in prnt_{G_1} \text{ and } v,w \in V_{G_1}\} \\
& \uplus \{(m_0+i,w) \mid (i,w) \in prnt_{G_1}, w \in V_{G_1} \text{ and } i \in m_1\} \\
& \uplus \{(v,n_0+i) \mid (v,i) \in prnt_{G_1}, v \in V_{G_1} \text{ and } i \in n_1\} \\
& \uplus \{(m_0+i,n_0+j) \mid (i,j) \in prnt_{G_1}, i \in m_1, \text{ and } j \in n_1\}.
\end{aligned}$$

Definition 6 (concrete link graph). *A* concrete link graph

$$F = (V_F, E_F, ctrl_F, link_F) : X \to Y$$

is a quadruple having an inner face X and an outer face Y, both finite subsets of $\mathcal{X}$, called respectivelly the inner and outer names of the link graph. F has finite sets $V_F \subset \mathcal{V}$ of nodes and $E_F \subset \mathcal{E}$ of edges, a control map $ctrl_F : V_F \to \mathcal{K}$ and a link map

$$link_F : X \uplus P_F \to E_F \uplus Y,$$

where $P_F \coloneqq \{(v,i) \mid i \in ar(ctrl_F(v))\}$ is the set of ports of F. Thus (v,i) is the ith port of node v. We shall call $X \uplus P_F$ the points *of F, and $E_F \uplus Y$ its* links*.*

The sets of points and the set of ports of a link l are defined by

$$points_F(l) \coloneqq \{p \mid link_F(p) = l\}, \quad ports_F(l) \coloneqq points_F(l) \setminus X.$$

An edge is *idle* if it has no points. Identities over name sets are defined by $\mathsf{id}_X = (\emptyset, \emptyset, \emptyset, \mathsf{Id}_X) : X \to X$.

Definition 7 (concrete bigraph with sharing). *A concrete bigraph*

$$F = (V_F, E_F, ctrl_F, prnt_F, link_F) : \langle k, X\rangle \to \langle m, Y\rangle$$

consists of a concrete place graph with sharing $F^P = (V_F, ctrl_F, prnt_F) : k \to m$ and a concrete link graph $F^L = (V_F, E_F, ctrl_F, link_F) : X \to Y$. We write the concrete bigraph with sharing as $F = (F^P, F^L)$.

Given a set of reaction rules, we refer to the configurations that a system may adopt as *states*. A bigraph with inner face ϵ is called *ground*.

Definition 8 (solid bigraph). *A bigraph is* solid *if these conditions hold:*

1. *no regions or outer names are idle (a region is called* idle *if it is empty);*
2. *no two sites or inner names are siblings;*
3. *the parent of every site is a node;*
4. *no outer name is linked to an inner name.*

Definition 9 (reaction rule). *A* reaction rule *is a pair*

$$\mathsf{R} = (R : m \to J, R' : m \to J),$$

sometimes written as $R \longrightarrow R'$*, where* R *is the* redex *and* R' *the* reactum*, and* R *is solid. The rule generates all the* ground reaction rules (r, r')*, where* $r = (R \otimes \mathsf{id}_Y) \circ d$ *and* $r' = (R' \otimes \mathsf{id}_Y) \circ d$ *for some discrete ground* parameter $d : \epsilon \to \langle m, Y \rangle$*. The* reaction relation $\longrightarrow\!\!\triangleright_{\mathsf{R}}$ *over ground bigraphs is defined by*

$$g \longrightarrow\!\!\triangleright_{\mathsf{R}} g' \text{ iff } g = D \circ r \text{ and } g' = D \circ r'$$

for some bigraph D *and some ground reaction rule* (r, r') *generated from* R*.*

Definition 10 (bigraphical reactive system (BRS)). *A* bigraphical reactive system *consists of a pair* $(\mathcal{B}, \mathcal{R})$*, where* $\mathcal{B}$ *is a set of ground bigraphs and* $\mathcal{R}$ *is a set of reaction rules defined over* $\mathcal{B}$*. It has a reaction relation*

$$\longrightarrow\!\!\triangleright_{\mathcal{R}} := \bigcup_{\mathsf{R} \in \mathcal{R}} \longrightarrow\!\!\triangleright_{\mathsf{R}},$$

which will be written $\longrightarrow\!\!\triangleright$ *when* $\mathcal{R}$ *is understood.*

Lastly, BigraphER supports *rule priorities* [1], i.e., a partial ordering on the reaction rules, where a rule of lower priority can be applied only if no rule of higher priority is applicable. Priorities are implemented by assigning each reaction rule to a *priority class*, where all classes are strictly ordered, and reaction rules in the same class have the same priority.

2 Extensions to Bigraphs

2.1 Probabilistic Bigraphs

Definition 11 (probabilistic reaction rule). *A* probabilistic reaction rule R *is a triple* (R, R', p)*, sometimes written* $R \xrightarrow{p} R'$*, where* (R, R') *is a reaction rule and* $p \in (0, 1]$ *is a probability. Similarly to Definition 9, it generates a set of ground reaction rules of the form* (r, r', p)*.*

Definition 12 (probabilistic bigraphical reactive system (PBRS)). *A* probabilistic bigraphical reactive system *consists of a pair* $(\mathcal{B}, \mathcal{R})$*, where* $\mathcal{B}$ *is a set of ground bigraphs and* $\mathcal{R}$ *is a set of probabilistic reaction rules defined over* $\mathcal{B}$*.*

Let g, g' *be ground bigraphs, and* $\{(r_i, r'_i, p_i)\}_{i=1}^n$ *a set of ground probabilistic reaction rules, where for each* r_i*, there exists a bigraph* D_i *such that* $g = D_i r_i$*. Let* $S = \{(r_i, r'_i, p_i) \mid g' = D_i r'_i\}$ *(for the same* D_i*), and*

$$s = \sum_{i=1}^{n} p_i.$$

Then the reaction relation is defined as

$$g \overset{p}{\longrightarrow\!\!\triangleright}_{\mathcal{R}} g' \text{ iff } S \neq \emptyset,$$

where

$$p = \frac{1}{s} \sum_{(r, r', p') \in S} p'.$$

2.2 Nondeterministic Bigraphs

Definition 13 (nondeterministic reaction rule). *Let A be a set of actions. A* nondeterministic reaction rule R *is a tuple (R, R', a, p), where (R, R', p) is a probabilistic reaction rule, and $a \in A$ is an action. We also define a* reaction reward function $r : A \to \mathbb{R}$ *that assigns a reward/cost to each action.*

Definition 14 (nondeterministic bigraphical reactive system (NBRS)). *A* nondeterministic bigraphical reactive system *consists of a pair $(\mathcal{B}, \mathcal{R})$, where $\mathcal{B}$ is a set of ground bigraphs and $\mathcal{R}$ is a set of nondeterministic reaction rules defined over $\mathcal{B}$.*

Let g, g' be ground bigraphs, $a \in A$ an action, and $\{(r_i, r'_i, a, p_i)\}_{i=1}^n$ a set of ground nondeterministic reaction rules with action a, where for each r_i, there exists a bigraph D_i such that $g = D_i r_i$. Let $S = \{(r_i, r'_i, a, p_i) \mid g' = D_i r'_i\}$ (for the same D_i), and

$$s = \sum_{i=1}^{n} p_i.$$

Then the reaction relation for action a is defined as

$$g \xrightarrow[r(a)]{p}\!\!\triangleright_a g' \text{ iff } S \neq \emptyset,$$

where

$$p = \frac{1}{s} \sum_{(r, r', a, p') \in S} p'.$$

BigraphER [4] supports defining predicates, i.e., bigraphs that are compared to every encountered state. We can associate a reward with each predicate, allowing us to assign rewards to states in a flexible and semantically meaningful way: the reward of a state is simply the sum of the rewards of all matching predicates (and 0 in case there are none).

3 Implementation

Example 1. Consider an MDP $(S, \overline{s}, A, P, L)$, where $S = \{s_0, s_1, s_2, s_3\}$, $\overline{s} = s_0$, $A = \{a, b, c\}$, and P, L defined as follows:

$$\begin{aligned} P(s_0, a) &= [s_1 \mapsto 1], \\ P(s_1, b) &= [s_0 \mapsto 0.7, s_1 \mapsto 0.3], \\ P(s_1, c) &= [s_2 \mapsto 0.5, s_3 \mapsto 0.5], \\ P(s_2, a) &= [s_2 \mapsto 1], \\ P(s_3, a) &= [s_3 \mapsto 1], \end{aligned} \qquad \begin{aligned} L(s_0) &= \{\mathit{initial}\}, \\ L(s_1) &= \emptyset, \\ L(s_2) &= \{\mathit{heads}\}, \\ L(s_3) &= \{\mathit{tails}\}. \end{aligned}$$

Furthermore, equip it with a reward structure (ρ, ι), where $\rho(s_2) = 3$, $\iota(s_1, b) = 1$, and both functions are zero everywhere else.

```
atomic ctrl S0 = 0;
atomic ctrl S1 = 0;
atomic ctrl S2 = 0;
atomic ctrl S3 = 0;

big initial = S0;
big heads = S2;
big tails = S3;

action a
  react toTemp = S0 -[1.]-> S1;
  react loopH = S2 -[1.]-> S2;
  react loopT = S3 -[1.]-> S3;
end

action b[1]
  react toInit = S1 -[0.7]-> S0
  react loop = S1 -[0.3]-> S1;
end

action c
  react toH = S1 -[0.5]-> S2;
  react toT = S1 -[0.5]-> S3;
end

begin nbrs
  init initial;
  rules = [ {toTemp, toInit,
    loop, toH, toT,
    loopH, loopT} ];
  preds = { initial, heads[3],
    tails };
end
```

Listing 1.1. BigraphER code.

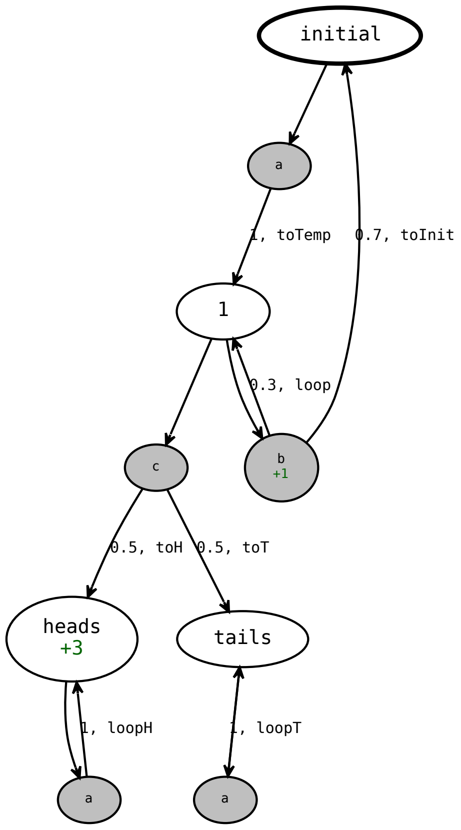
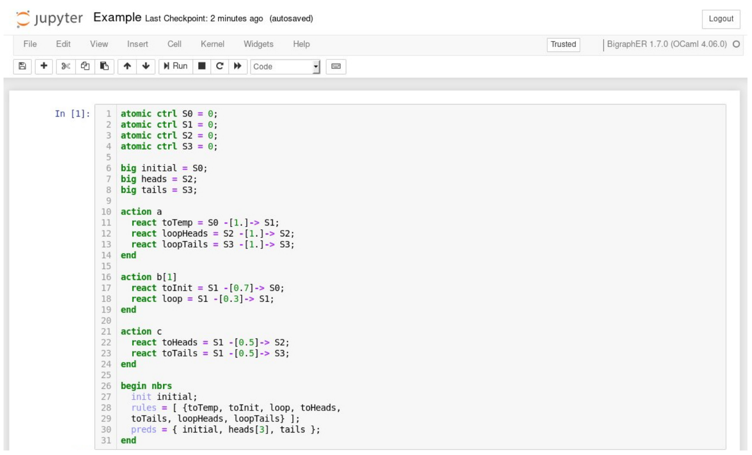


Fig. 2. The full transition system.

The MDP can be represented as an NBRS with BigraphER code in Listing 1.1. Furthermore, the BigraphER's visualisation of the (automatically generated) transition system is in Fig. 2.

More specifically, reaction rule probabilities are represented as floating-point numbers inside the arrows (e.g. `-[0.7]->`), each action encompasses its reaction rules with **`action`** `actionName` and **`end`**. Rewards can be added to actions simply by inserting an integer enclosed in squared brackets after the action name (e.g. **`action`** `b[1]`). Lastly, predicate rewards have the same format, but are listed on the **`preds`** line of the **`begin nbrs`**—**`end`** block (e.g. **`preds`** `= {heads[3]};`).

Visually, each state is represented by a white ellipse (the starting state is highlighted with a thicker border), and each action (per state) is a grey ellipse. Text inside state ellipses lists all predicates that are satisfied by that state. Edges from actions to states are labelled with both the probability and the name of the reaction rule. Transitions from either a fully generated transition system or a simulation, labels from predicates, and state/transition rewards can be exported to the corresponding PRISM plain text formats.


jupyter
Example Last Checkpoint: 2 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
BigraphER 1.7.0 (OCaml 4.06.0)
Run
Code
In [1]:

```
atomic ctrl S0 = 0;
atomic ctrl S1 = 0;
atomic ctrl S2 = 0;
atomic ctrl S3 = 0;

big initial = S0;
big heads = S2;
big tails = S3;

action a
  react toTemp = S0 -[1.]-> S1;
  react loopHeads = S2 -[1.]-> S2;
  react loopTails = S3 -[1.]-> S3;
end

action b[1]
  react toInit = S1 -[0.7]-> S0;
  react loop = S1 -[0.3]-> S1;
end

action c
  react toHeads = S1 -[0.5]-> S2;
  react toTails = S1 -[0.5]-> S3;
end

begin nbrs
  init initial;
  rules = [ {toTemp, toInit, loop, toHeads,
  toTails, loopHeads, loopTails} ];
  preds = { initial, heads[3], tails };
end
```

Fig. 3. The BigraphER Jupyter interface with syntax highlighting.

4 A Case Study in Autonomous Agents

4.1 Collecting Objects in a Grid

Our first scenario is concerned with collecting objects in a grid-like two-dimensional world. The agent starts in the southwest corner of the map (which we call *home*), explores the environment until it collects a specified number of objects, and returns home (while possibly collecting more objects along the way). The model tracks which cells of the grid have been explored, and only the unexplored cells have a constant probability *p* of rewarding the agent with an object. The desired number of objects can be easily changed, and the map can be customised to make some transitions unavailable (i.e. add walls), as long as the agent can still return home.

```
ctrl Cell = 1;
ctrl Agent = 0;
ctrl Directions = 0;
ctrl Object = 0;

atomic ctrl North = 1;
atomic ctrl East = 1;
atomic ctrl West = 1;
atomic ctrl South = 1;

atomic ctrl Visited = 1;
atomic ctrl Unvisited = 1;
```

Listing 1.2. Controls.

We begin with a list of controls in Listing 1.2. On the right hand side of each equal sign is the number of ports that each node of that control has, while the **`atomic`** keyword specifies the nodes that cannot contain other nodes. `Cells` represents discrete locations. They are linked to either `Visited` or `Unvisited` (initially, all except one cells are unvisited). `Cells` always contain `Directions`, and one `Cell` contains an `Agent`. The `Agent` may contain an `Object`. In order to represent multiple objects, it is convenient to put the second `Object` inside the first, and so on. `Directions` contains a subset of the four nodes, corresponding to the four possible directions of travel: `North`, `East`, `West`, `South`. We connect neighbouring `Cells` by linking the `West` node of one `Cell` to the `East` node of another (or `North` to `South`).

```
big home = Cell{v}.(Directions.(North{v1}
                              | East{v2})
                  | Agent);
big goal = Agent.Object;
big initial = Visited{v}
           || Unvisited{u}
           # bottom left
           || Cell{v}.(Directions.(North{a} | East{b})
                    | Agent.1)
           # top left
           || Cell{u}.Directions.(East{c} | South{a})
           # bottom right
           || Cell{u}.Directions.(North{d} | West{b})
           # top right
           || Cell{u}.Directions.(West{c} | South{d});
```

Listing 1.3. Predicates and the initial state.

Next, we define two predicates: `home` and `goal` (see Listing 1.3). The former states that the agent is home if it is in the southwest corner of the grid, while the latter specifies a goal of collecting one object. A larger number of objects can be set by nesting `Objects` inside each other, e.g., `Agent.Object.Object`. Additionally, Listing 1.3 defines the initial 2×2 grid, as described previously.

```
action north
  react goNorthToVisited =
       Visited{v}
    || Cell{v}.(Directions.(North{b} | id) | Agent)
    || Cell{v}.Directions.(South{b} | id)
    -[1.0]->
       Visited{v}
    || Cell{v}.Directions.(North{b} | id)
    || Cell{v}.(Directions.(South{b} | id) | Agent)
      @ [0, 2, 1];
  react goNorthToUnvisited =
       Visited{v} || Unvisited{u}
    || Cell{v}.(Directions.(North{b} | id) | Agent)
    || Cell{u}.Directions.(South{b} | id)
    -[1.0-p]->
       Visited{v} || Unvisited{u}
    || Cell{v}.Directions.(North{b} | id)
    || Cell{v}.(Directions.(South{b} | id) | Agent)
      @ [0, 2, 1];
  react northObject =
       Visited{v} || Unvisited{u}
    || Cell{v}.(Directions.(North{b} | id) | Agent)
    || Cell{u}.Directions.(South{b} | id)
    -[p]->
       Visited{v} || Unvisited{u}
    || Cell{v}.Directions.(North{b} | id)
    || Cell{v}.(Directions.(South{b} | id)
              | Agent.Object)
      @ [0, 2, 1];
end
```

Listing 1.4. The action of going north.

Then, we have an action for each of the four directions, each with three reaction rules: one for going to an already visited cell, one for visiting a cell for the first time, and one for visiting a cell for the first time and finding an object. We will use the rules for going north in Listing 1.4 as an example. When going to an already visited cell, the `Agent` simply traverses a link between the `North` node of its previous `Cell` and the corresponding `South` node. As this is the only reaction rule for this situation, its probability is 1. When going to an unvisited node, we collect an `Object` with probability p, or make the transition without getting an `Object` with probability $1 - p$. Regardless, the `Cell` changes its link from `Unvisited` to `Visited`.

```
react stayHome =
  Cell{v}.(Directions.(North{v1} | East{v2}) | goal)
  -[1.0]->
  Cell{v}.(Directions.(North{v1} | East{v2}) | goal);
```

Listing 1.5. Reaction rule to stay home if a required number of objects is acquired.

Lastly, we have three actions for when the agent acquires the specified number of objects: two for going home west or south (since these are the two directions necessary and sufficient to reach home), and one for staying home. The two actions for going home have two reaction rules each, depending on whether the destination cell is visited or not. They are just like the rules in Listing 1.4, except every mention of `Agent` is replaced with `goal`. The rule to stay home simply says that if the agent has achieved the goal, and is home (characterised by a `Cell` with only two directions: `North` and `East`), then nothing should change (see Listing 1.5). To make the agent start heading home as soon as enough objects are collected, all five reaction rules related to going and staying home are in a higher priority class.

4.2 Layers of Abstraction

Our second model explores two important concepts in modelling the topology of space: nesting and uncertainty. Nesting is the idea that spaces can be inside other spaces: rooms inside floors, floors inside buildings, buildings inside streets, etc. A key characteristic of nested spaces is that there is a separate action (or several) taking you from the outer space to the inner space or vice versa (e.g., using a door to enter or leave a room). In the rest of the paper, whenever we discuss one space inside another, we will refer to the inner space as the *room*.

The other concept explored in this section is uncertainty about the topology itself, i.e., is the door open or closed? The implementation of this idea is tied to that of rooms: the inside of a room is not defined until the outer cell encompassing the room (i.e., the area where the door is located) is entered. Then, the contents of the room are chosen from a finite set of possibilities with a set probability distribution. After that, the room stays fixed. This design decision supports the reality of missions with a reasonably short duration: if the agent finds a locked door, it should not keep revisiting the door hoping for it to suddenly be open.

As the ideas of collecting objects and tracking visited cells have been explored already, they are dropped from this model to simplify the implementation. The goal can simply be defined as moving from one location to another. We add one extra control, `Node`. While `Cell` can be imagined as a discrete patch of space (e.g., $1\,\mathrm{m}^2$), `Node` can be thought of as a room. While we stick to two layers (corridors and rooms) in this example, rooms can be freely nested: a room can contain any number of rooms, each of which can contain more rooms.

```
big initial = /x /y /z (
```

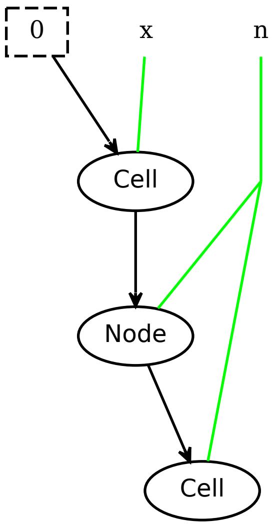


Fig. 4. The bigraphical structure behind nesting.

```
   Cell{x}.(Directions.East{a} | Agent)
|| Cell{y}.(Directions.(East{b} | West{a})
          | Node{n}.1)
|| Cell{z}.Directions.West{b});
```

Listing 1.6. The initial map: three consecutive cells, the midmost of which has an inner room.

In order to impement nesting, a `Cell` contains a `Node`, which can have any number of `Cells` of its own. Since after entering a room, you end up in a specific place inside the room (next to the door), we also need to link up the `Node` with one of the `Cells` it contains (see Fig. 4). Both `Cells` and `Nodes` support having a single link for exactly this purpose. Thus, most of the `Cell` ports are not linked to anything, which is implemented via closures (e.g. `/x Cell{x}`). The initial map in Listing 1.6 has a corridor of three cells, with the agent on the west-most cell, and a door to a room in the middle cell. One moves in and out of a room via a pair of actions with reaction rules pictured in Fig. 5.

```
action room
  react openDoor =
    Cell{w}.(Agent | Node{n}.1 | id)
    -[p]-> /x /y /z /a /b /c /d
    Cell{w}.(Agent | Node{n}.(
      # top left
        Cell{n}.Directions.(East{a} | South{b})
      # top right
      | Cell{x}.Directions.(West{a} | South{c})
```

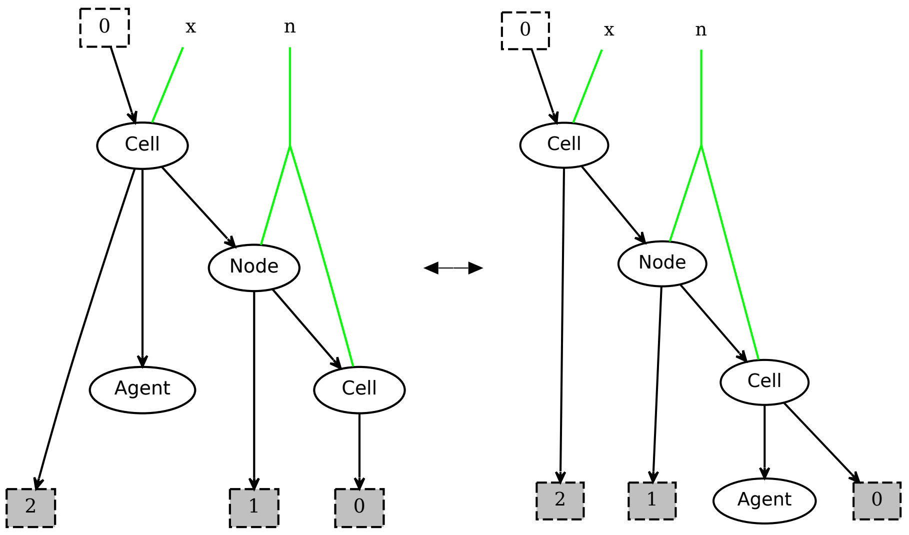


Fig. 5. Left to right: the reaction rule to go inside; right to left: the reaction rule to go outside.

```
      # bottom left
      | Cell{y}.Directions.(North{b} | East{d})
      # bottom right
      | Cell{z}.Directions.(North{c} | West{d}))
      | id);
  react closedDoor =
    Cell{w}.(Agent | Node{n}.1 | id)
    -[1.0-p]-> /x /y /z /b /c /d
    Cell{w}.(Agent | Node{n}.(
      # top left
        Cell{n}.Directions.(South{b})
      # top right
      | Cell{x}.Directions.(South{c})
      # bottom left
      | Cell{y}.Directions.(North{b} | East{d})
      # bottom right
      | Cell{z}.Directions.(North{c} | West{d}))
      | id);
end
```

Listing 1.7. Two ways to generate a room, leaving one door either open or closed.

```
action north
  react goNorth =
       Cell{n1}.(Directions.(North{b} | id)
              | Agent | id)
    || Cell{n2}.(Directions.(South{b} | id) | id)
    -[1.0]->
       Cell{n1}.(Directions.(North{b} | id) | id)
    || Cell{n2}.(Directions.(South{b} | id)
              | Agent | id);
end
```

Listing 1.8. The action of going north.

The room is a 2×2 grid of `Cells`, which is generated when the `Agent` gets to a `Cell` with an empty `Node` (represented by `Node{n}.1`). In order for this to happen before any other action can take place, the two reaction rules in Listing 1.7 are put on a higher priority class. With probability p, the generated room has each `Cell` linked to its two neighbours; and with probability $1 - p$, the top two `Cells` are not linked , possibly forcing the agent to choose a different path. Finally, we have four actions, each with a single reaction rule with probability 1, corresponding to the four possible directions of travel. The reaction rules are very similar to those of our previous example, see Listing 1.8.

5 Conclusion

References

1. Baeten, J.C.M., Bergstra, J.A., Klop, J.W., Weijland, W.P.: Term-rewriting systems with rule priorities. Theor. Comput. Sci. **67**(2&3), 283–301 (1989). https://doi.org/10.1016/0304-3975(89)90006-6, https://doi.org/10.1016/0304-3975(89)90006-6
2. Sevegnani, M.: Bigraphs with sharing and applications in wireless networks. Ph.D. thesis, University of Glasgow, UK (2012), http://theses.gla.ac.uk/3742/
3. Sevegnani, M., Calder, M.: Bigraphs with sharing. Theor. Comput. Sci. **577**, 43–73 (2015). https://doi.org/10.1016/j.tcs.2015.02.011, https://doi.org/10.1016/j.tcs.2015.02.011
4. Sevegnani, M., Calder, M.: BigraphER: Rewriting and analysis engine for bigraphs. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 494–501. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_27, https://doi.org/10.1007/978-3-319-41540-6_27