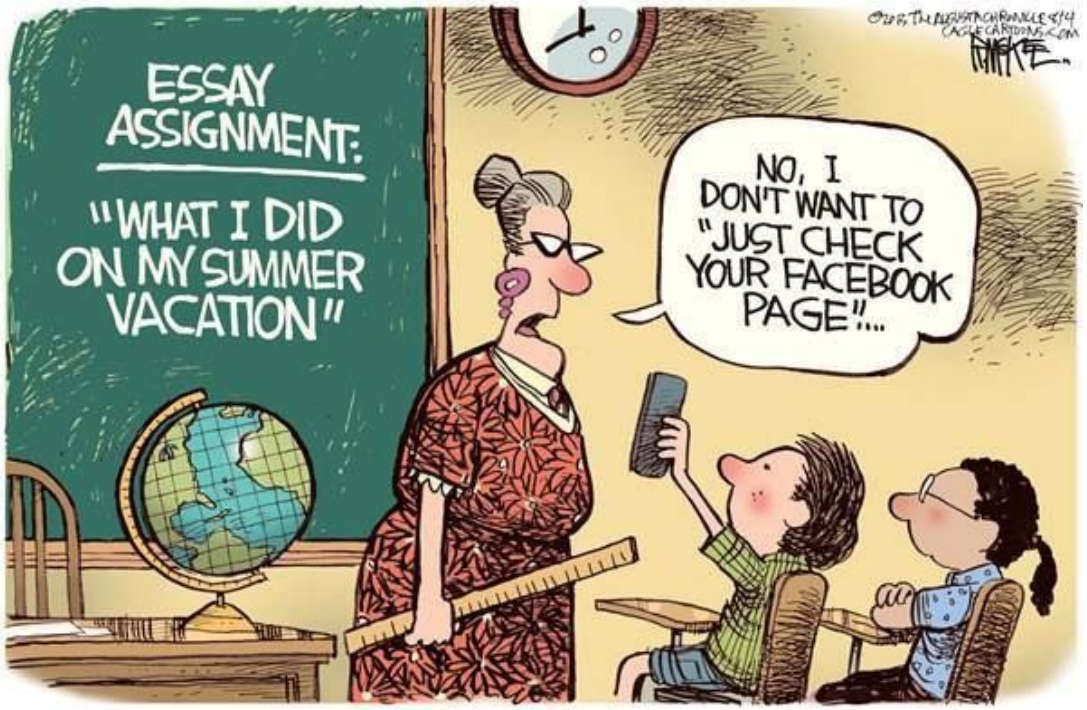
Nondeterministic Bigraphs and Their Use in Modelling Movement

Paulius Dilkas

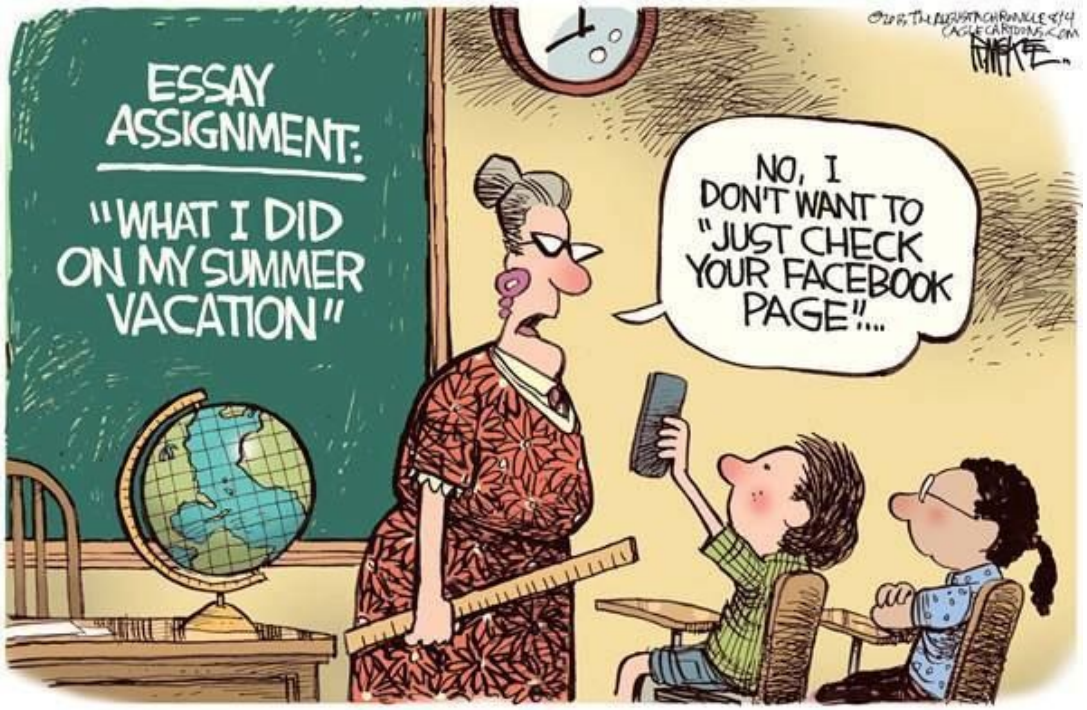
Formal Analysis, Theory and Algorithms

16th October 2018

Overview

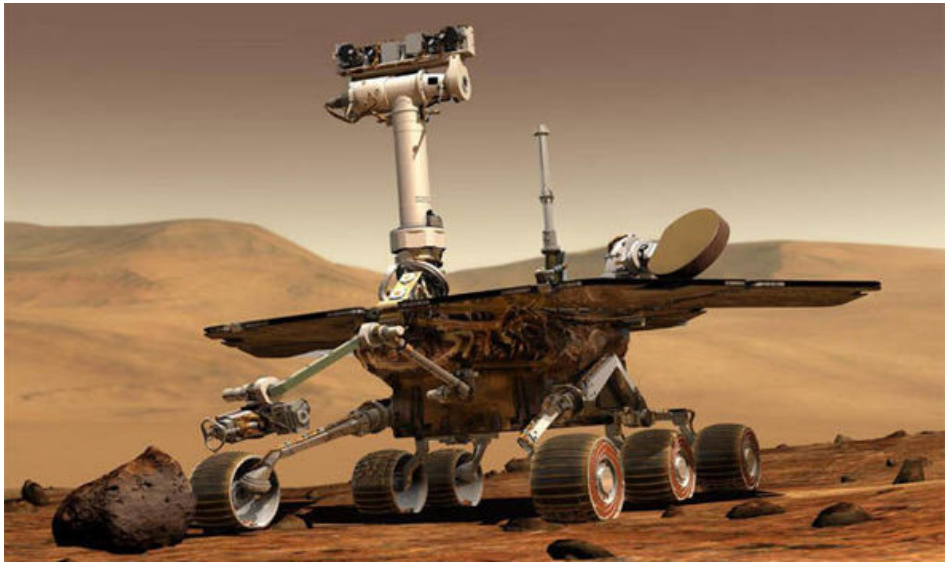

ESSAY ASSIGNMENT:
"WHAT I DID ON MY SUMMER VACATION"
NO, I DON'T WANT TO "JUST CHECK YOUR FACEBOOK PAGE"...

Overview

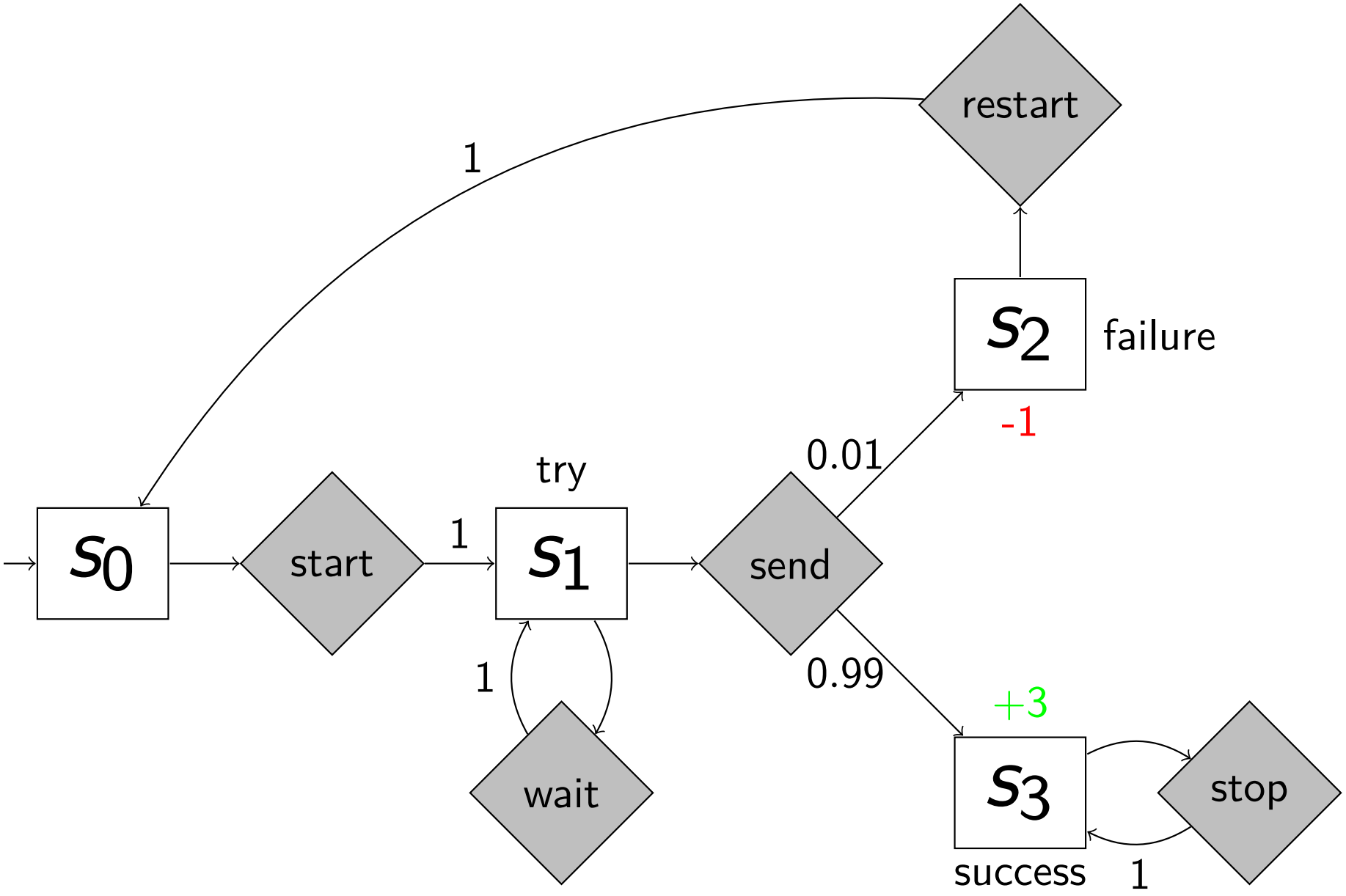


- How to use bigraphs to model situations with decisions and rewards?
- Examples in modelling movement
- A full picture of bigraphs: from code to visualisations to inner workings

Motivation



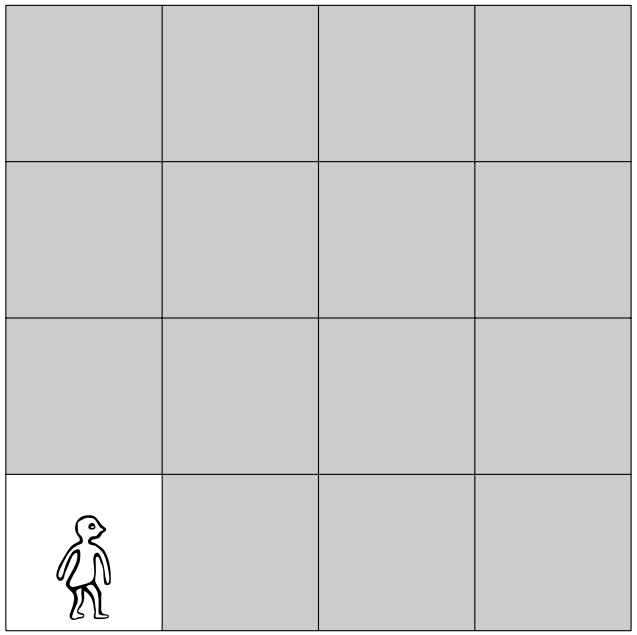
Markov Decision Process



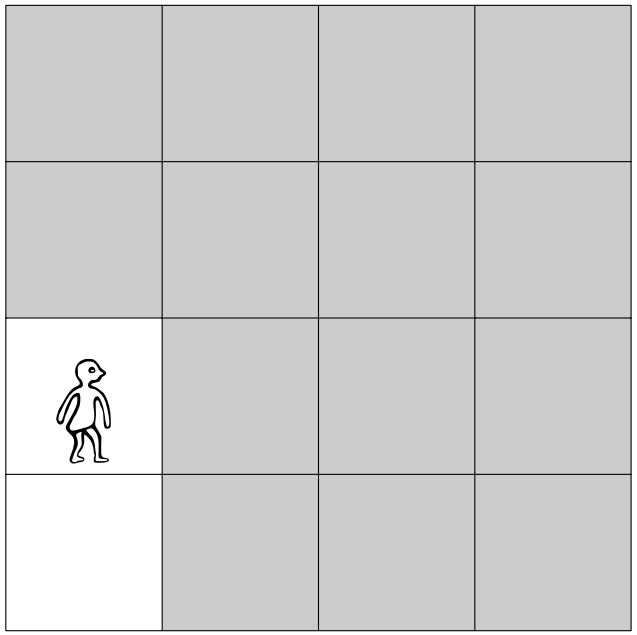
Collecting Objects in a Grid

- Each cell is either visited or unvisited.
- When entering an unvisited cell, with probability p the agent receives an object.
- Once a set number of objects is collected, the agent heads home.

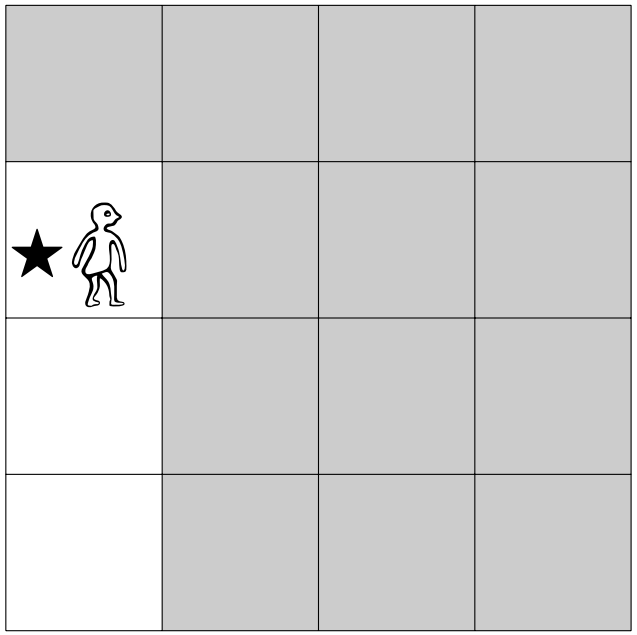
Collecting Objects in a Grid



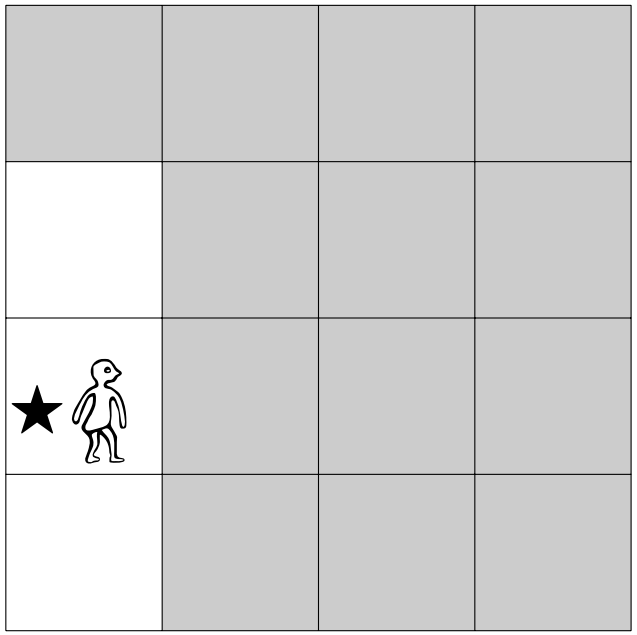
Collecting Objects in a Grid



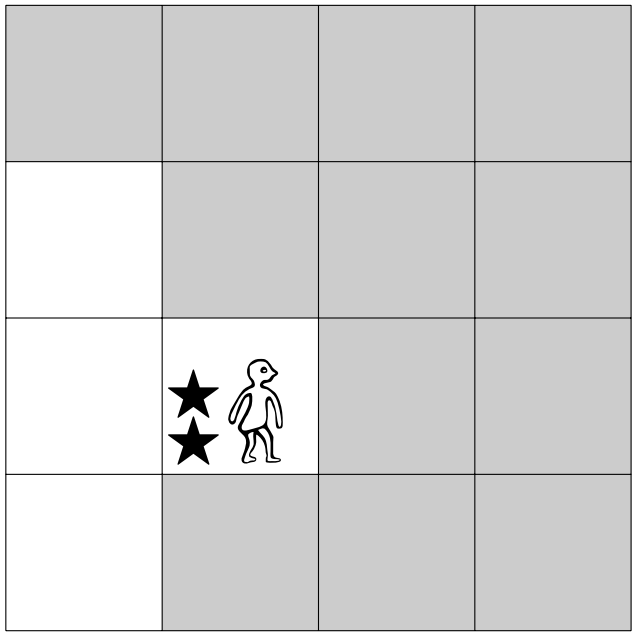
Collecting Objects in a Grid



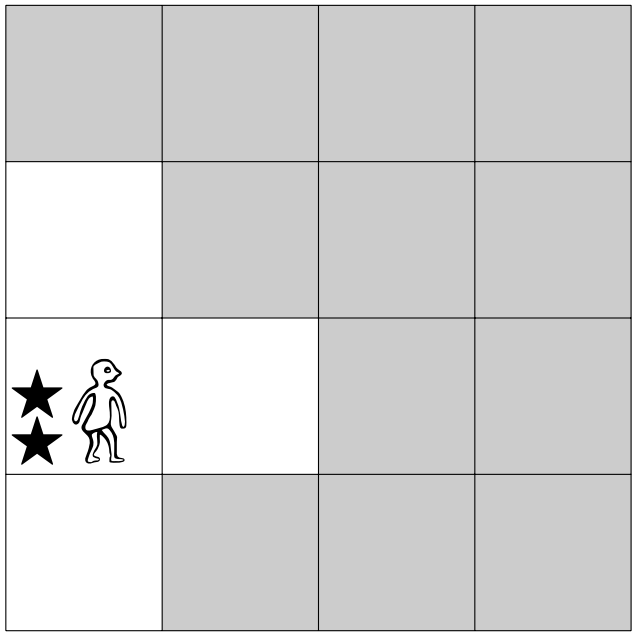
Collecting Objects in a Grid



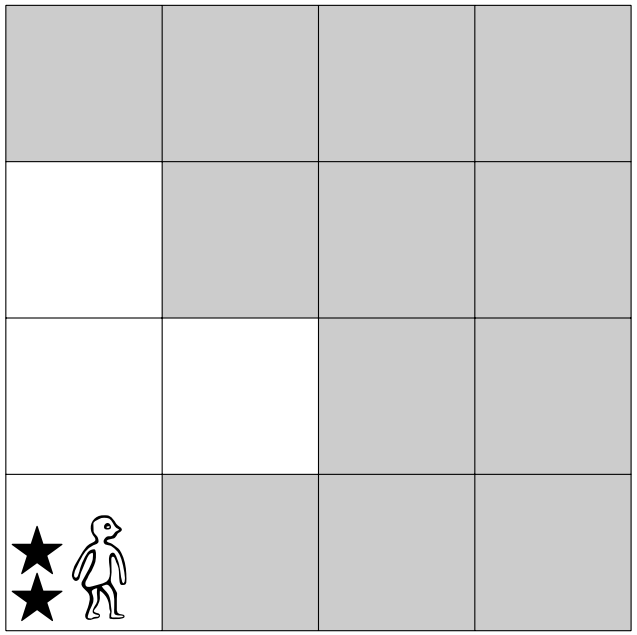
Collecting Objects in a Grid



Collecting Objects in a Grid



Collecting Objects in a Grid



A High Level View

- Controls (types of nodes)

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`
- Predicates (properties to check)
  - `goal`: collected the required number of objects
  - `home`: is in the southwest corner of the grid

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`
- Predicates (properties to check)
  - `goal`: collected the required number of objects
  - `home`: is in the southwest corner of the grid
- Reaction rules (how the state changes)

A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`
- Predicates (properties to check)
  - `goal`: collected the required number of objects
  - `home`: is in the southwest corner of the grid
- Reaction rules (how the state changes)
  - Grouped into actions by direction
  - Different rules for going to visited and unvisited cells

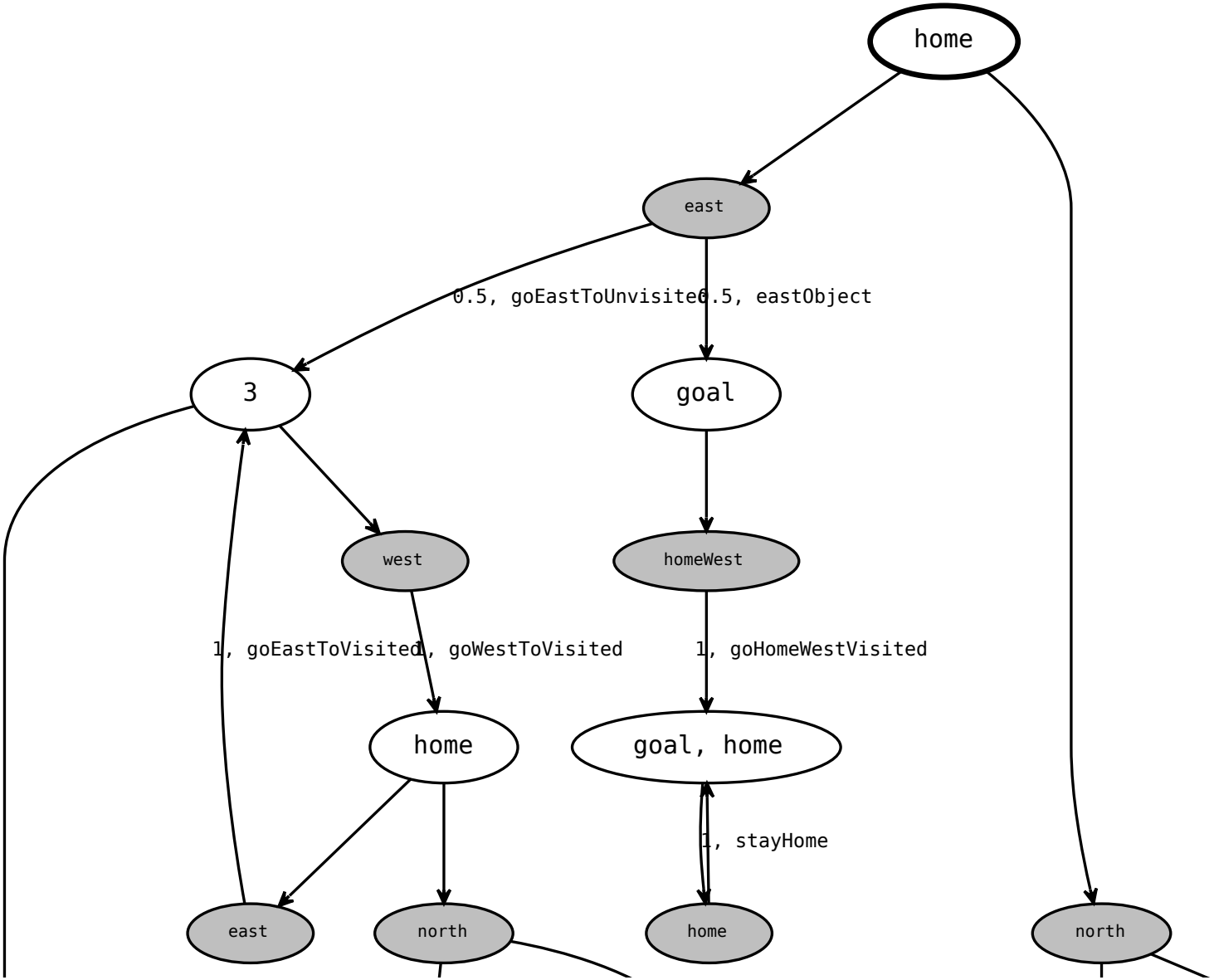
A High Level View

- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`
- Predicates (properties to check)
  - `goal`: collected the required number of objects
  - `home`: is in the southwest corner of the grid
- Reaction rules (how the state changes)
  - Grouped into actions by direction
  - Different rules for going to visited and unvisited cells
  - Priority 1: going/staying home (5 rules)

A High Level View

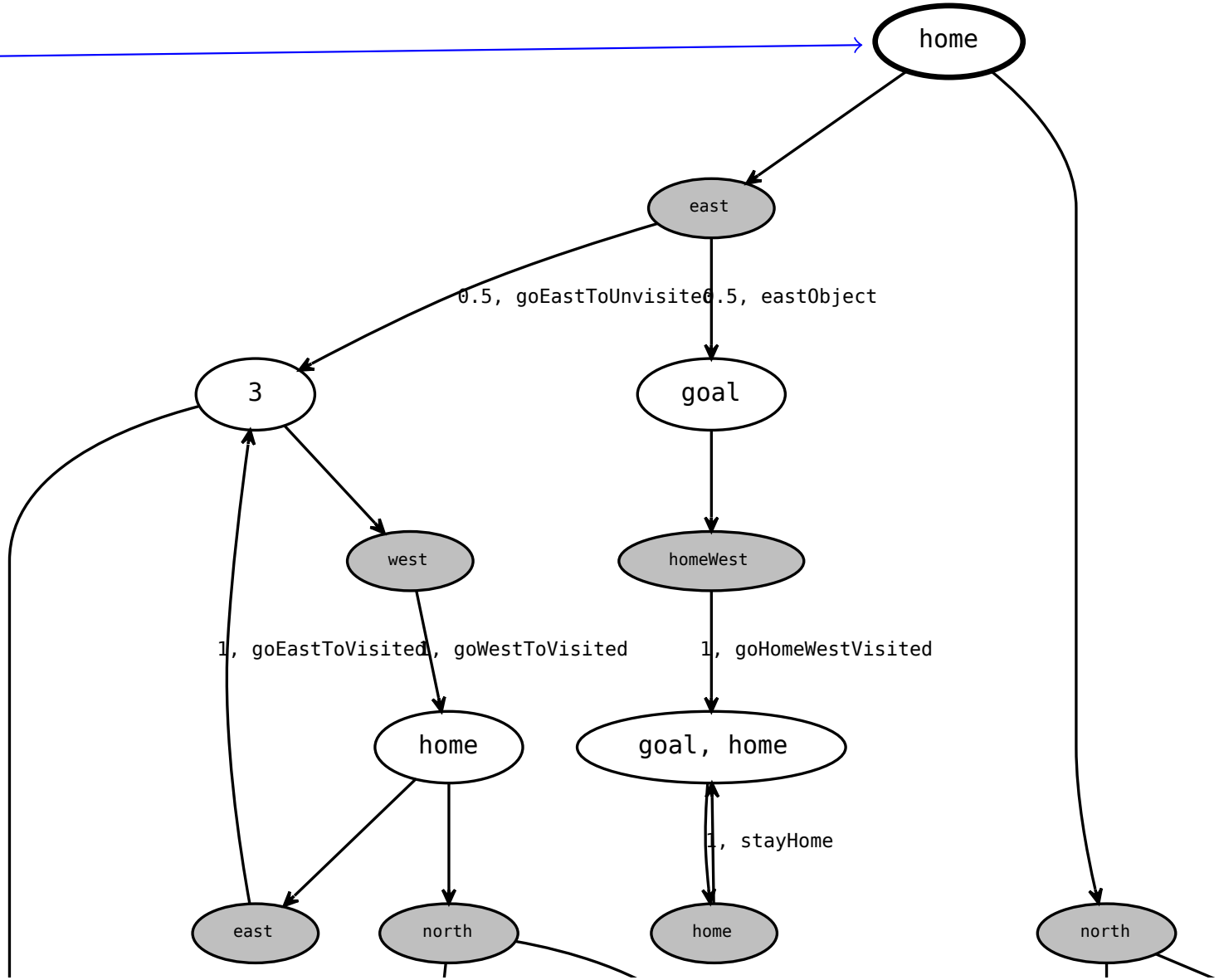
- Controls (types of nodes)
  - `Agent, Cell, Directions, Object`
  - `North, East, West, South`
  - `Visited, Unvisited`
- Predicates (properties to check)
  - `goal`: collected the required number of objects
  - `home`: is in the southwest corner of the grid
- Reaction rules (how the state changes)
  - Grouped into actions by direction
  - Different rules for going to visited and unvisited cells
  - Priority 1: going/staying home (5 rules)
  - Priority 2: 3 rules for each direction
    - ★ visited
    - ★ unvisited
    - ★ unvisited + object

Transition System



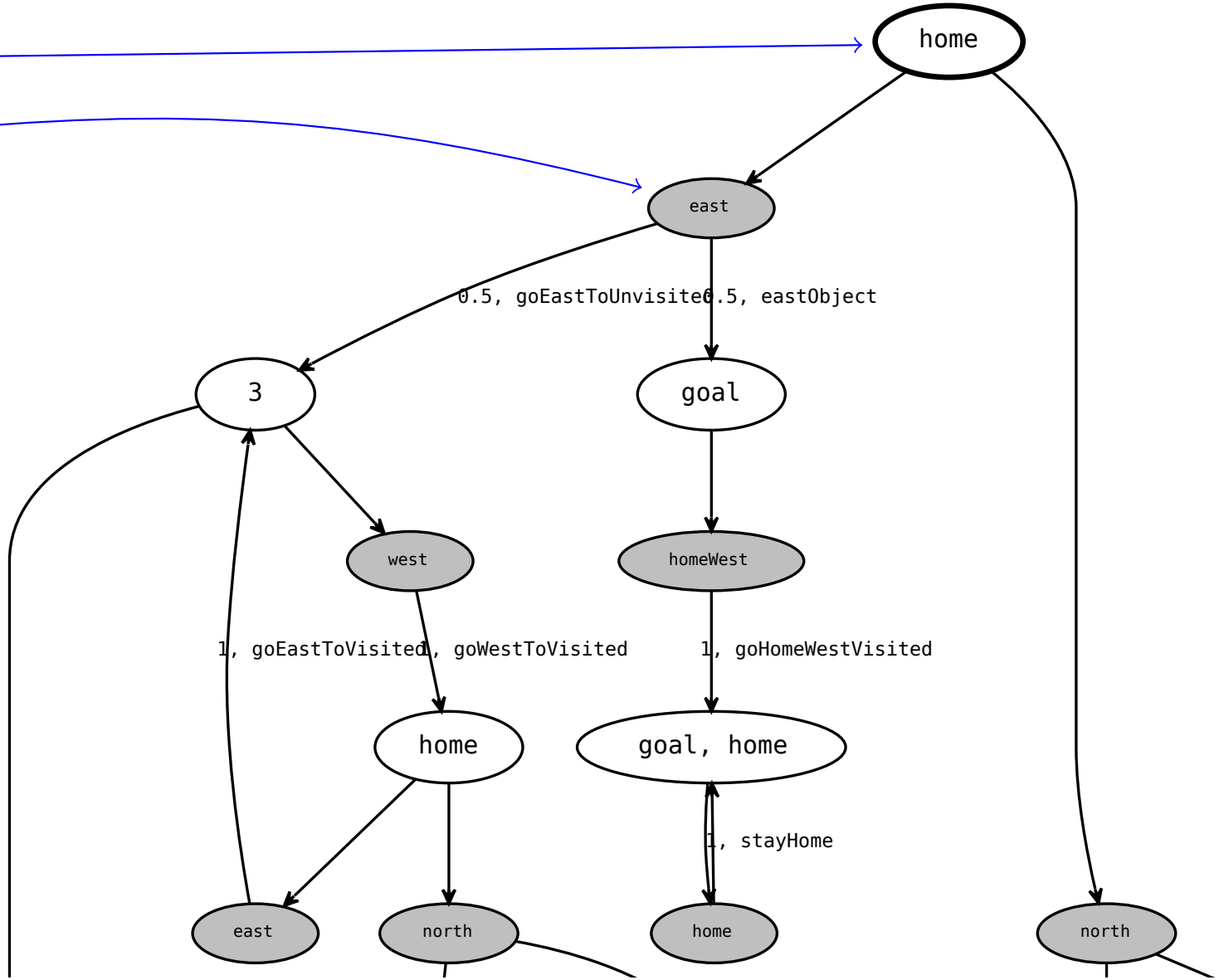
Transition System

- States



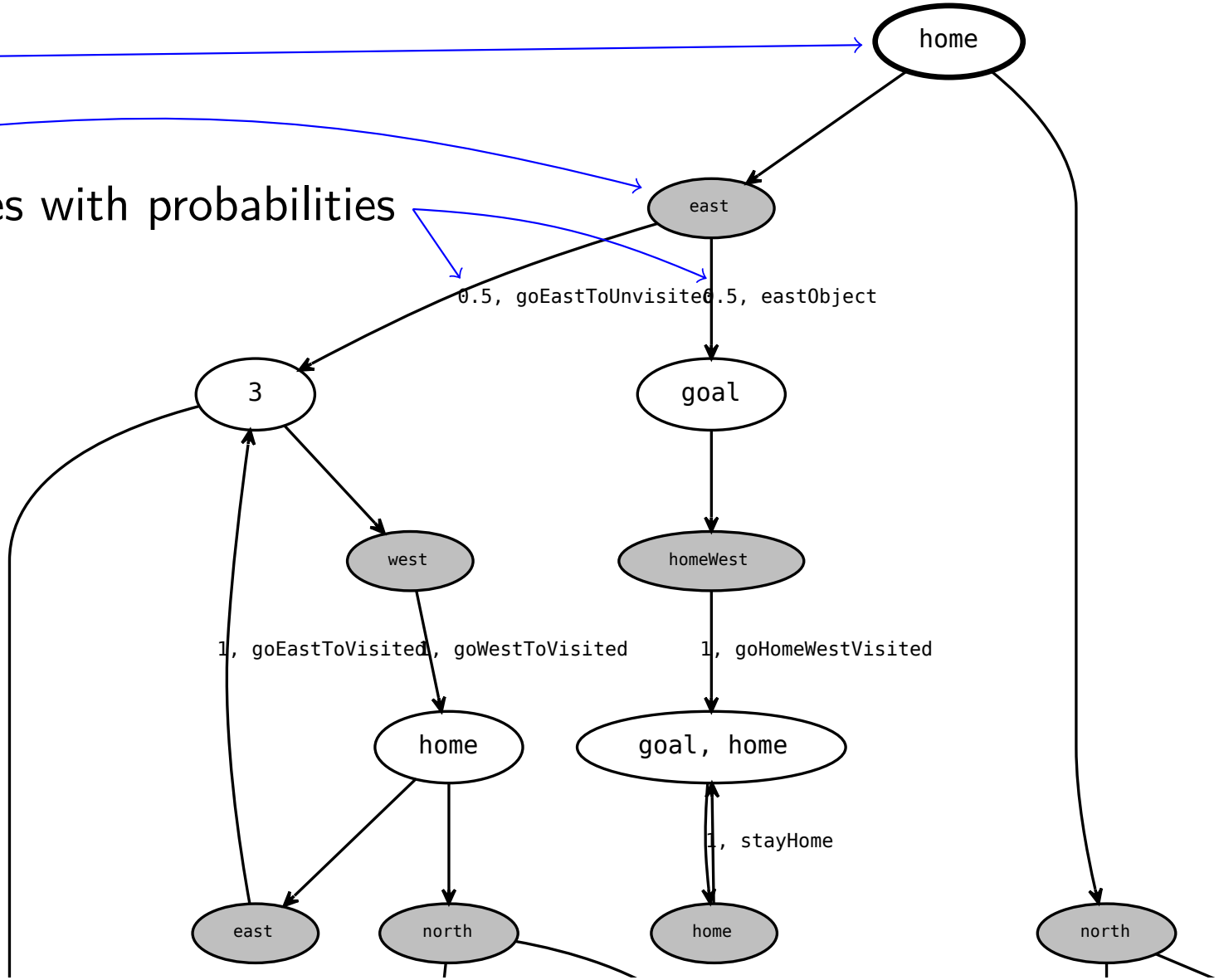
Transition System

- States
- Actions



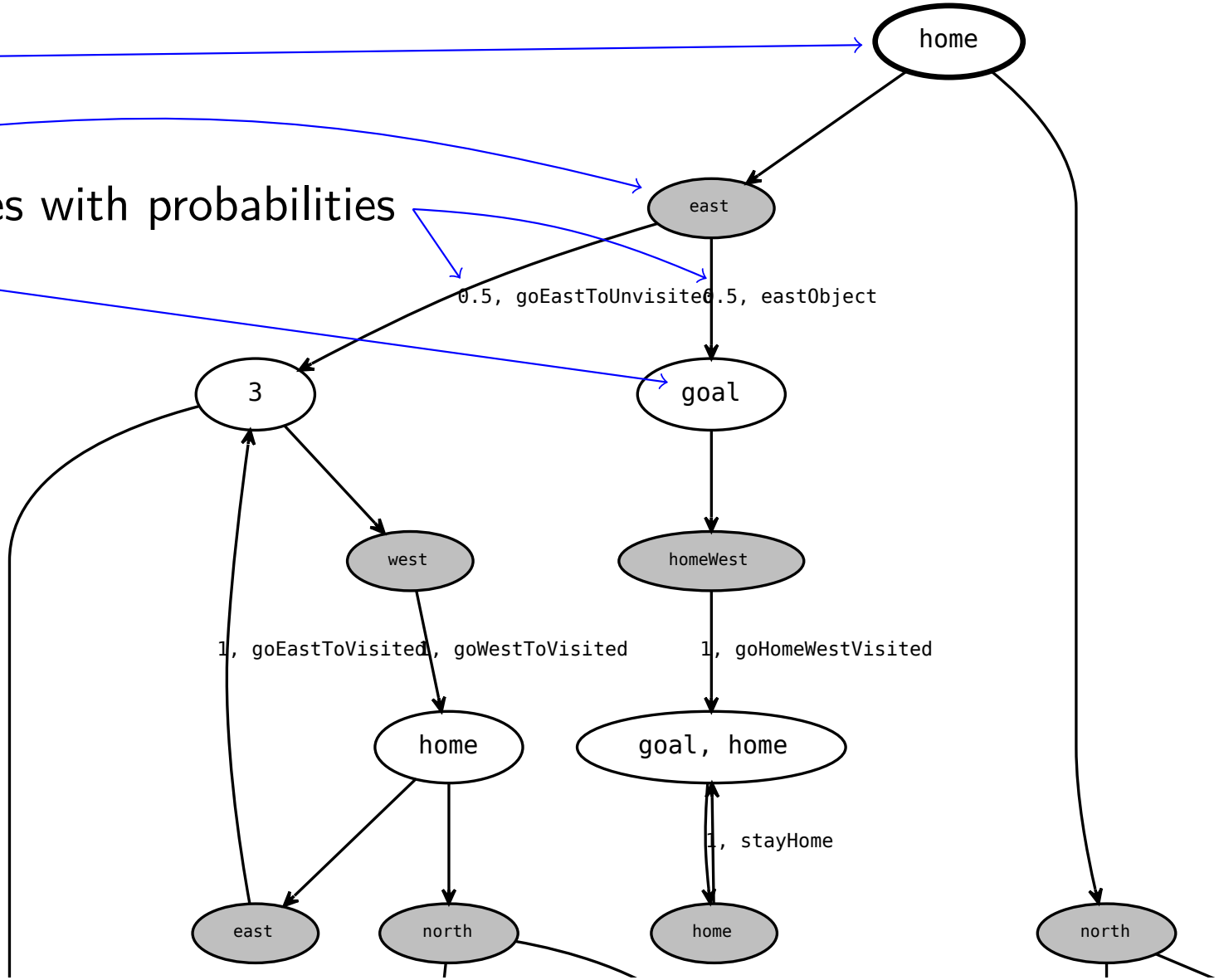
Transition System

- States
- Actions
- Reaction rules with probabilities



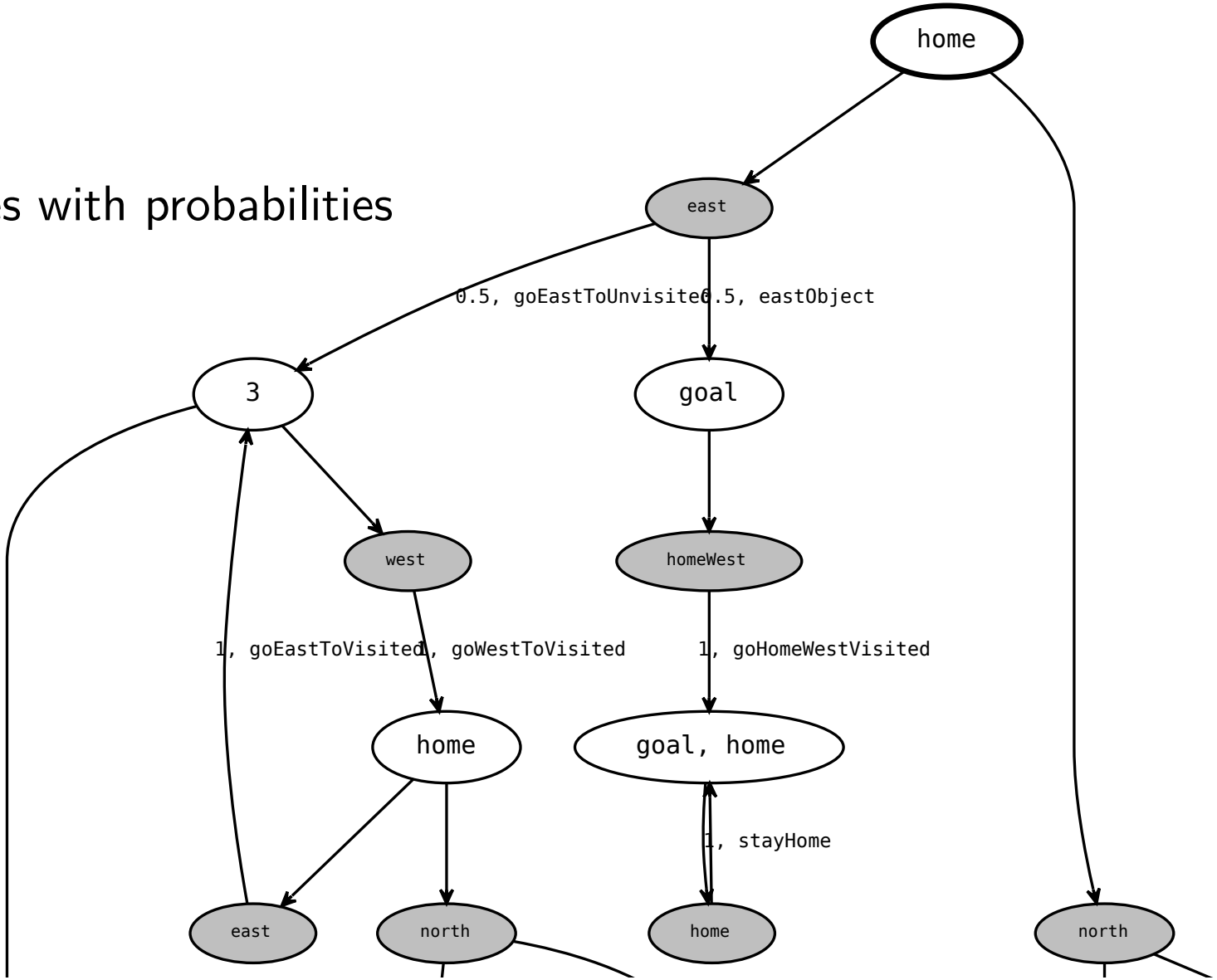
Transition System

- States
- Actions
- Reaction rules with probabilities
- Predicates

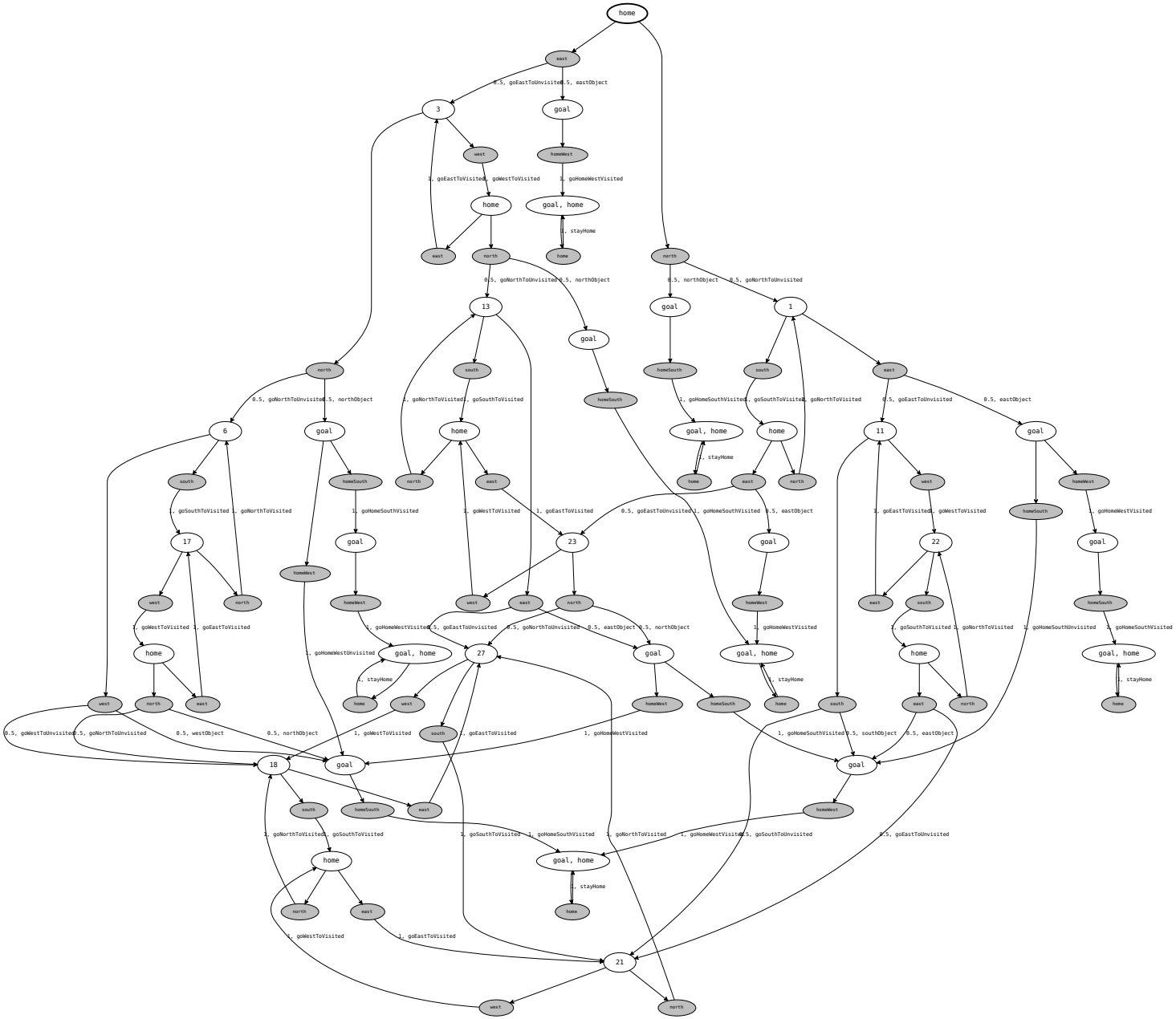


Transition System

- States
- Actions
- Reaction rules with probabilities
- Predicates



Transition System



The Workflow

- Start with an initial state

The Workflow

- Start with an initial state
- Find all applicable reaction rules (from the highest non-empty priority class)
  - Priorities and actions are orthogonal concepts

The Workflow

- Start with an initial state
- Find all applicable reaction rules (from the highest non-empty priority class)
  - Priorities and actions are orthogonal concepts
- Normalise probabilities per action

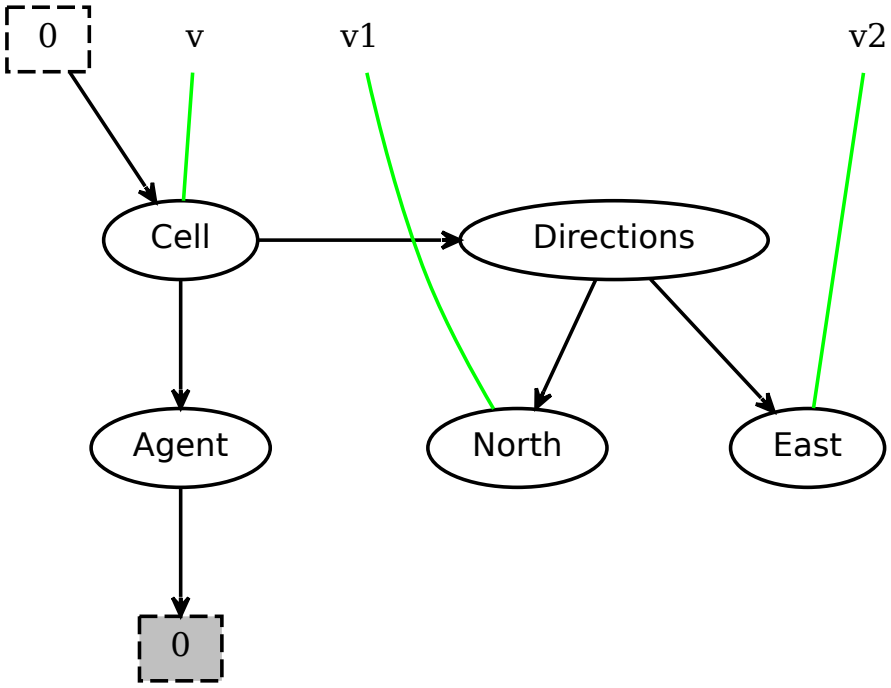
The Workflow

- Start with an initial state
- Find all applicable reaction rules (from the highest non-empty priority class)
  - Priorities and actions are orthogonal concepts
- Normalise probabilities per action
  - Caveat: one rule can sometimes be applied in multiple ways
  - In that case, all outcomes are equally likely

The Workflow

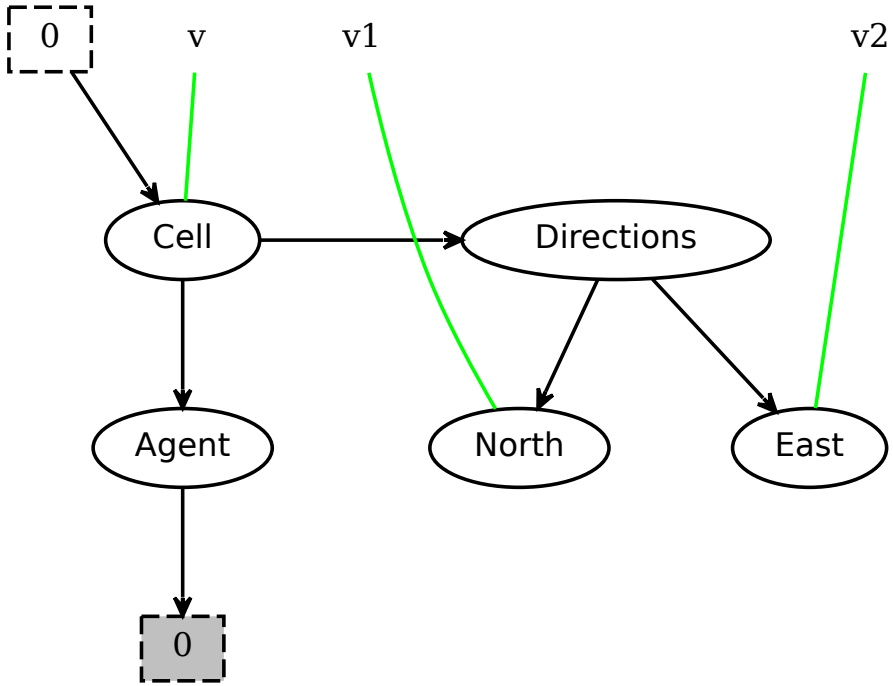
- Start with an initial state
- Find all applicable reaction rules (from the highest non-empty priority class)
  - Priorities and actions are orthogonal concepts
- Normalise probabilities per action
  - Caveat: one rule can sometimes be applied in multiple ways
  - In that case, all outcomes are equally likely
- Either:
  - Breadth first search to generate the full transition system
  - Or select the next state randomly for a simulation

Bigraphs



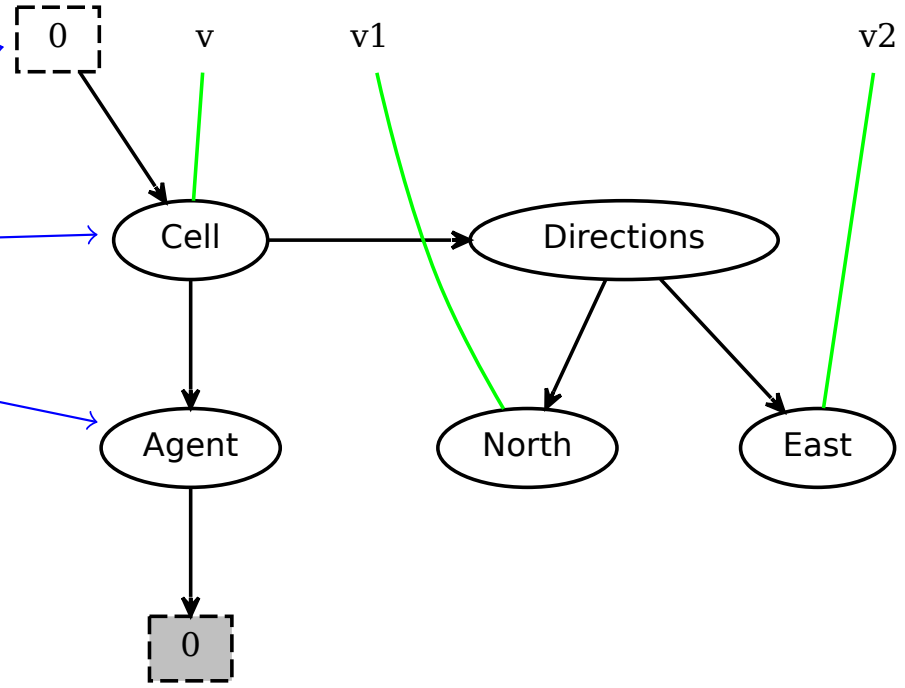
Bigraphs

- Region



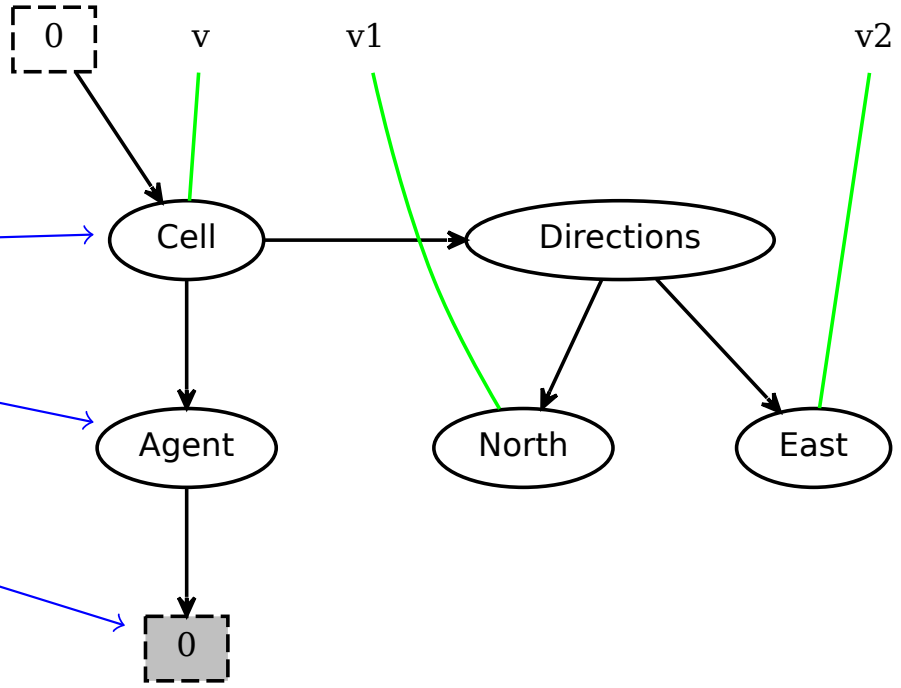
Bigraphs

- Region
- Nodes



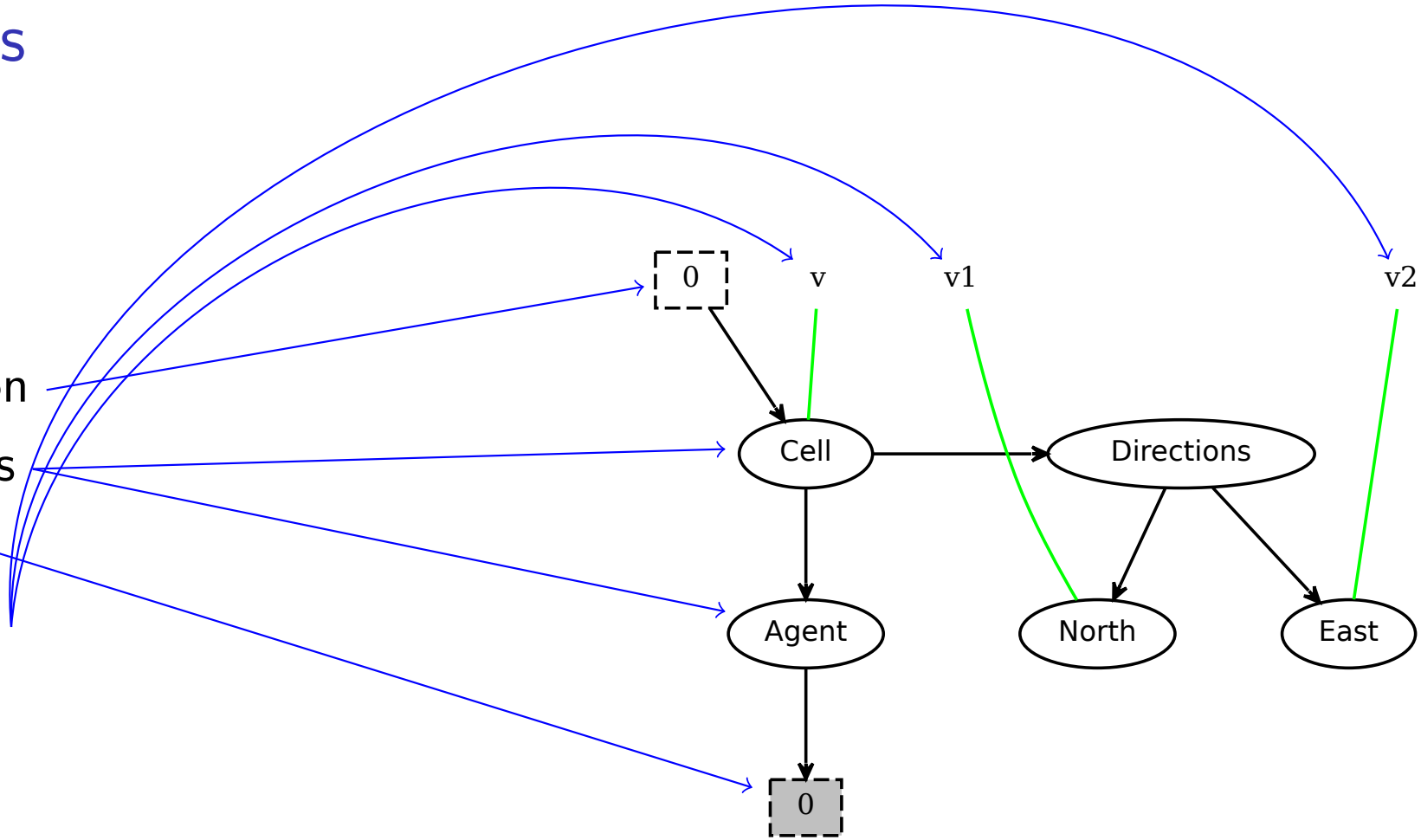
Bigraphs

- Region
- Nodes
- Site

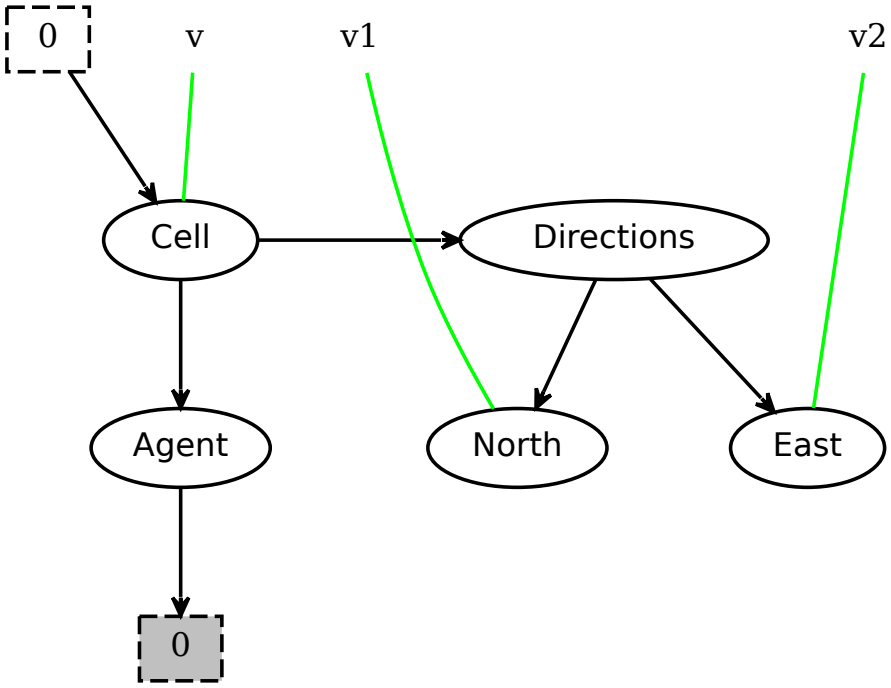


Bigraphs

- Region
- Nodes
- Site
- Links



Bigraphs



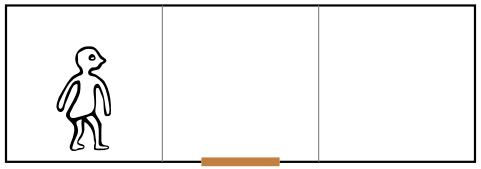

```
big home = Cell{v}.(Directions.(North{v1}
                                | East{v2})
                  | Agent);
```

Initial State

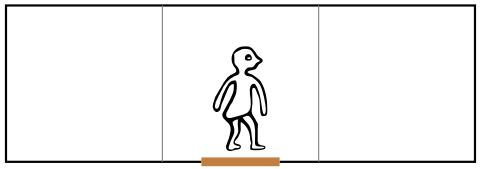
```
big initial = Visited{v}
            || Unvisited{u}
            # bottom left
            || Cell{v}.(Directions.(North{a}
                                  | East{b})
                      | Agent.1)
            # top left
            || Cell{u}.Directions.(East{c}
                            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

A Tale of Schrödinger's Wall...

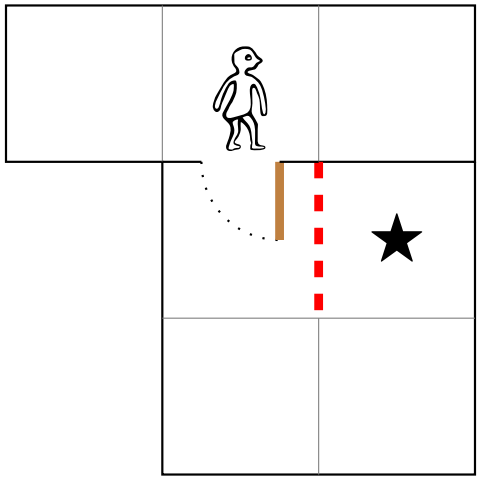
A Tale of Schrödinger's Wall...



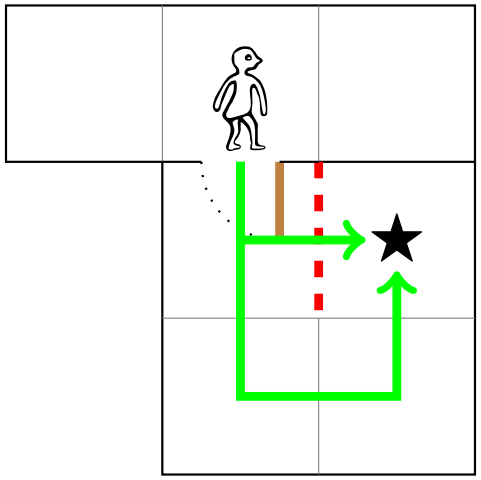
A Tale of Schrödinger's Wall...



A Tale of Schrödinger's Wall...



A Tale of Schrödinger's Wall...



A High Level View

- Controls
  - `Agent`, `Cell`, `Directions`, `Goal`, `Node`
  - `North`, `East`, `West`, `South`
  - no `Object`, no `Visited/Unvisited`

A High Level View

- Controls
  - `Agent`, `Cell`, `Directions`, `Goal`, `Node`
  - `North`, `East`, `West`, `South`
  - no `Object`, no `Visited/Unvisited`
- Reaction rules
  - Priority 1: generating the room (2 rules in 1 action)

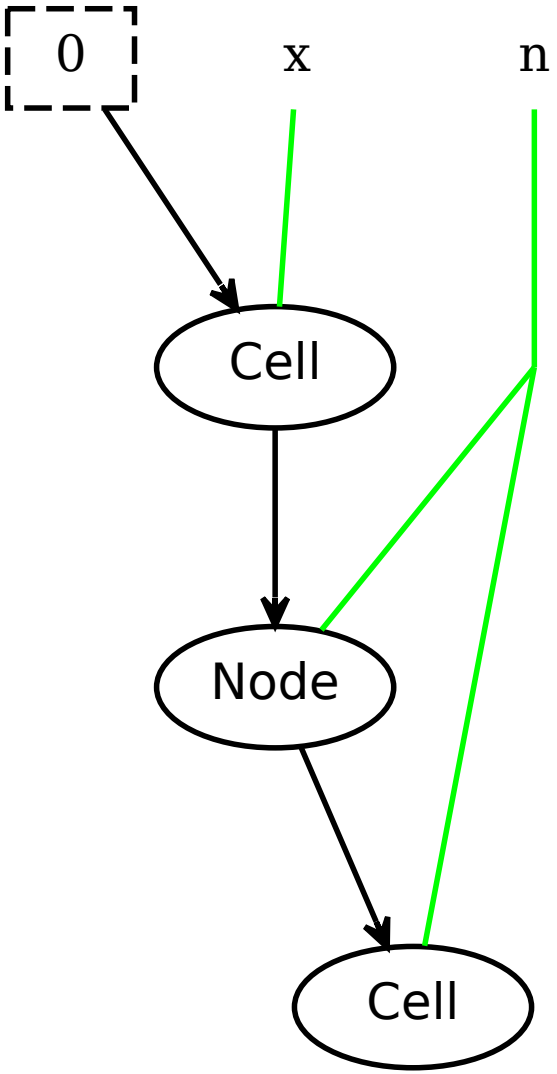
A High Level View

- Controls
  - `Agent`, `Cell`, `Directions`, `Goal`, `Node`
  - `North`, `East`, `West`, `South`
  - no `Object`, no `Visited/Unvisited`
- Reaction rules
  - Priority 1: generating the room (2 rules in 1 action)
  - Priority 2: movement in 6 directions (including going in/out)
    - ★ each rule in a separate action

A High Level View

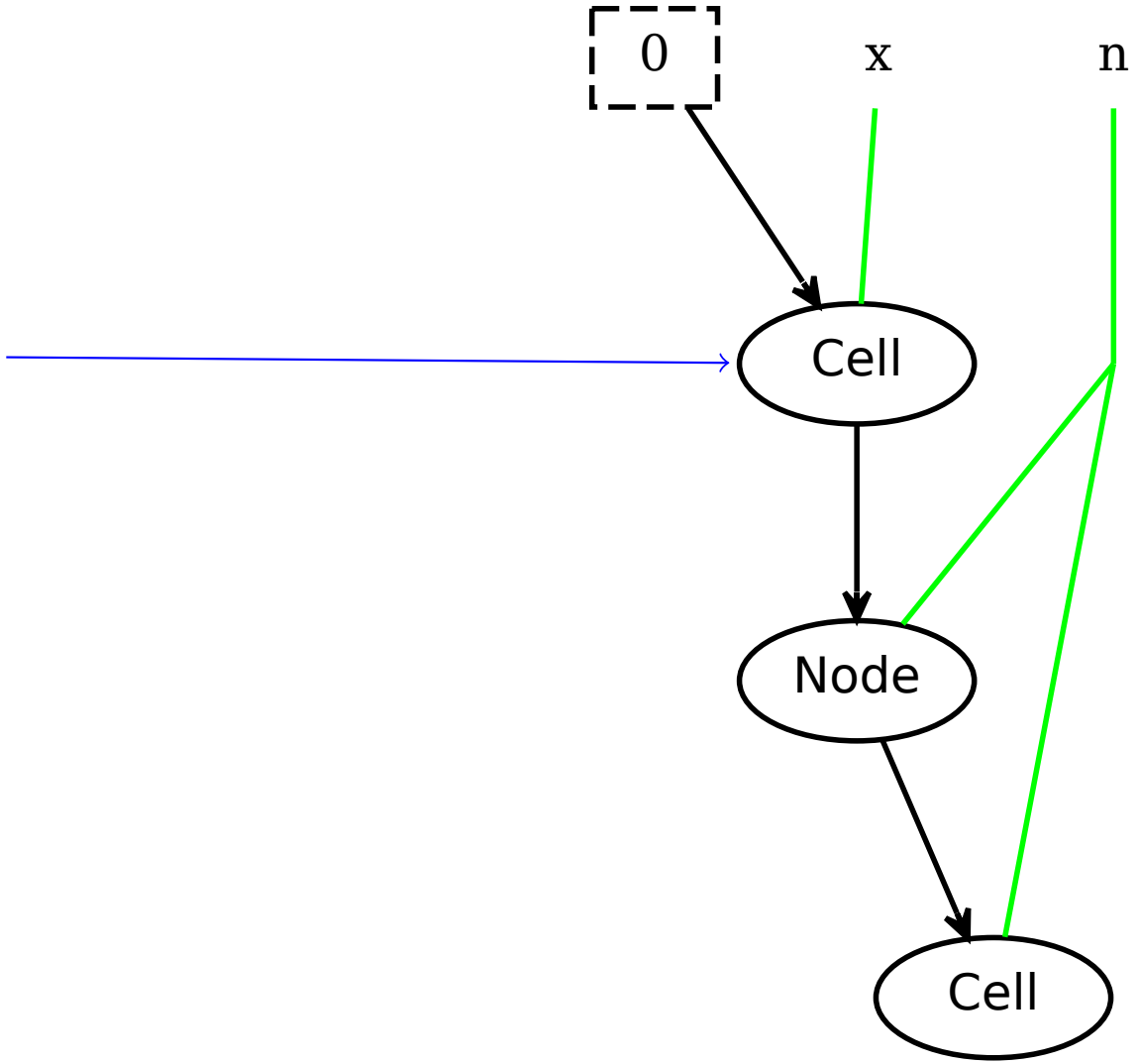
- Controls
  - `Agent`, `Cell`, `Directions`, `Goal`, `Node`
  - `North`, `East`, `West`, `South`
  - no `Object`, no `Visited/Unvisited`
- Reaction rules
  - Priority 1: generating the room (2 rules in 1 action)
  - Priority 2: movement in 6 directions (including going in/out)
    - ★ each rule in a separate action
- Predicate
  - are `Agent` and `Goal` in the same cell?

The Main Idea



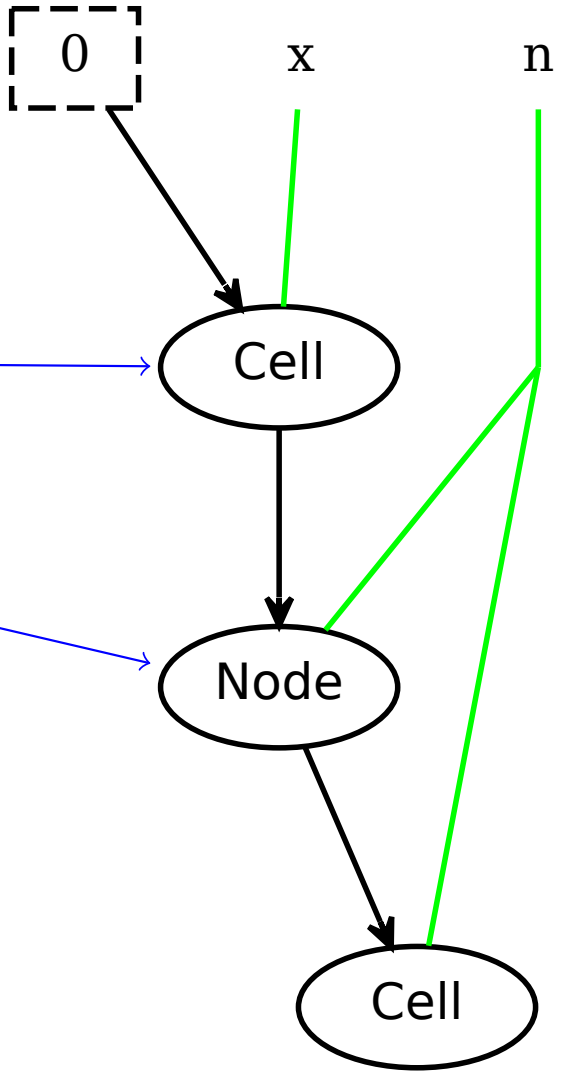
The Main Idea

- Outside the door



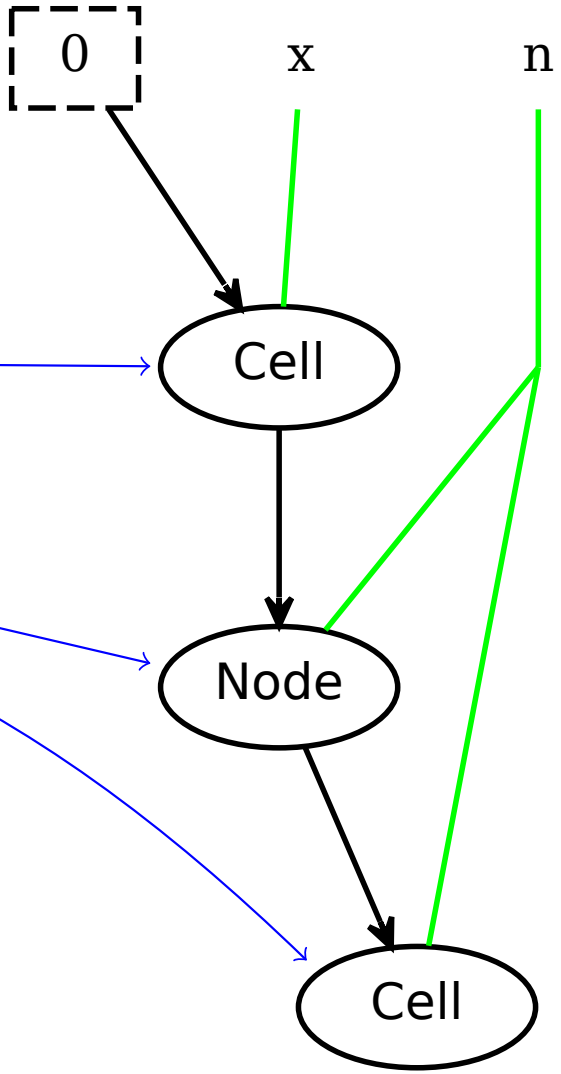
The Main Idea

- Outside the door
- The room



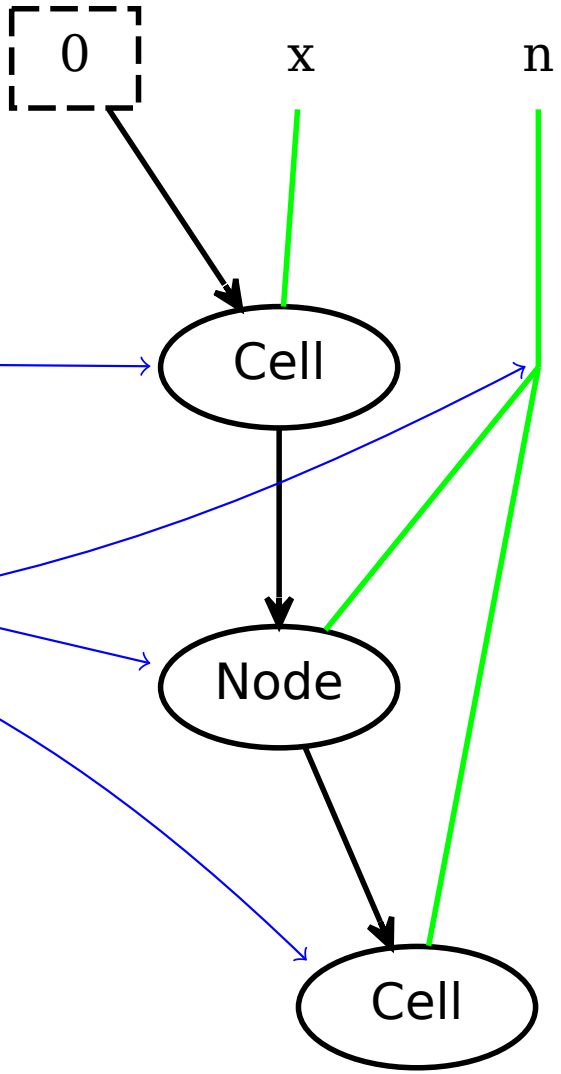
The Main Idea

- Outside the door
- The room
- Inside the door

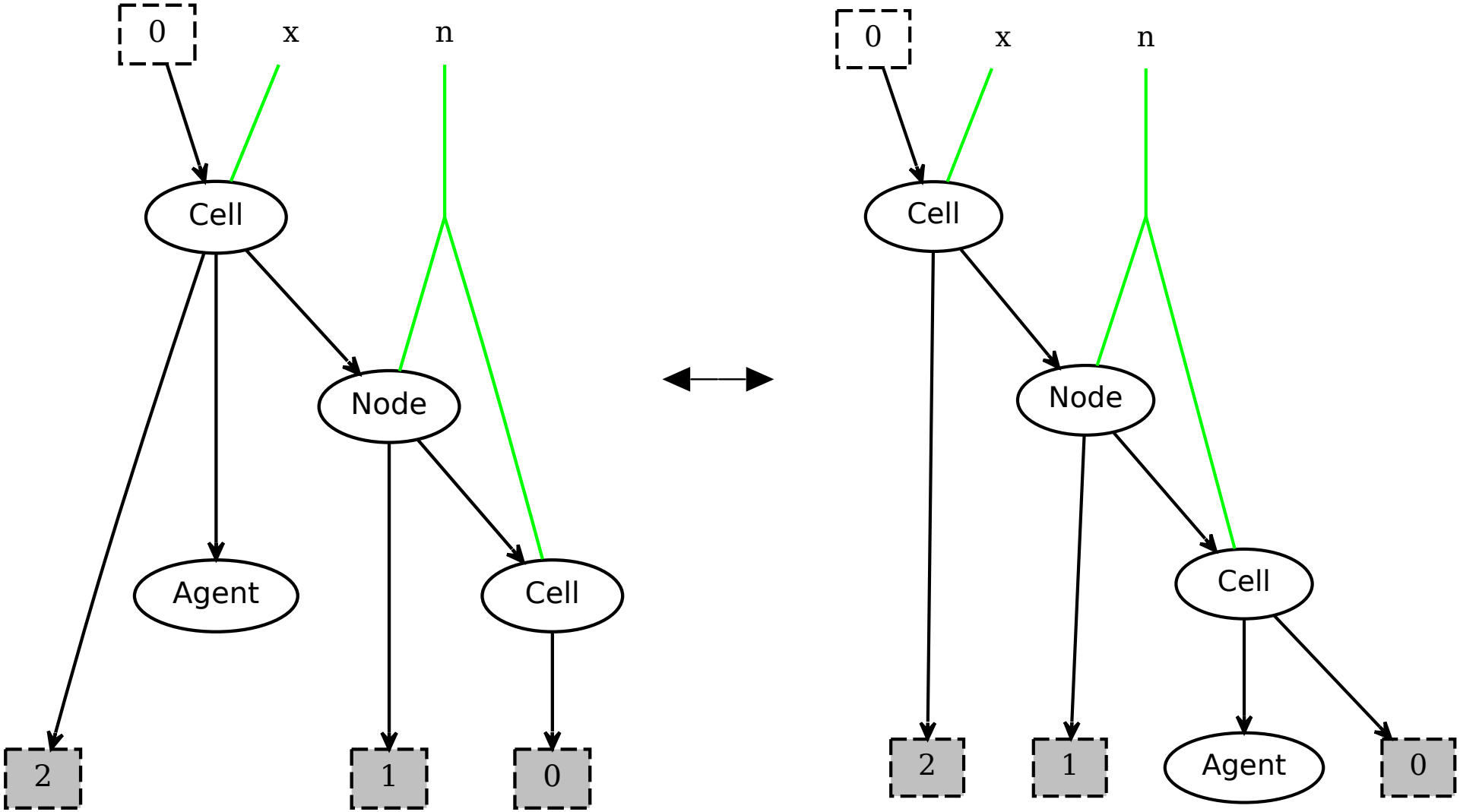


The Main Idea

- Outside the door
- The room
- Inside the door
- Which cell is closest to the door?



Reaction Rules for Entering/Leaving a Room



Entering/Leaving a Room

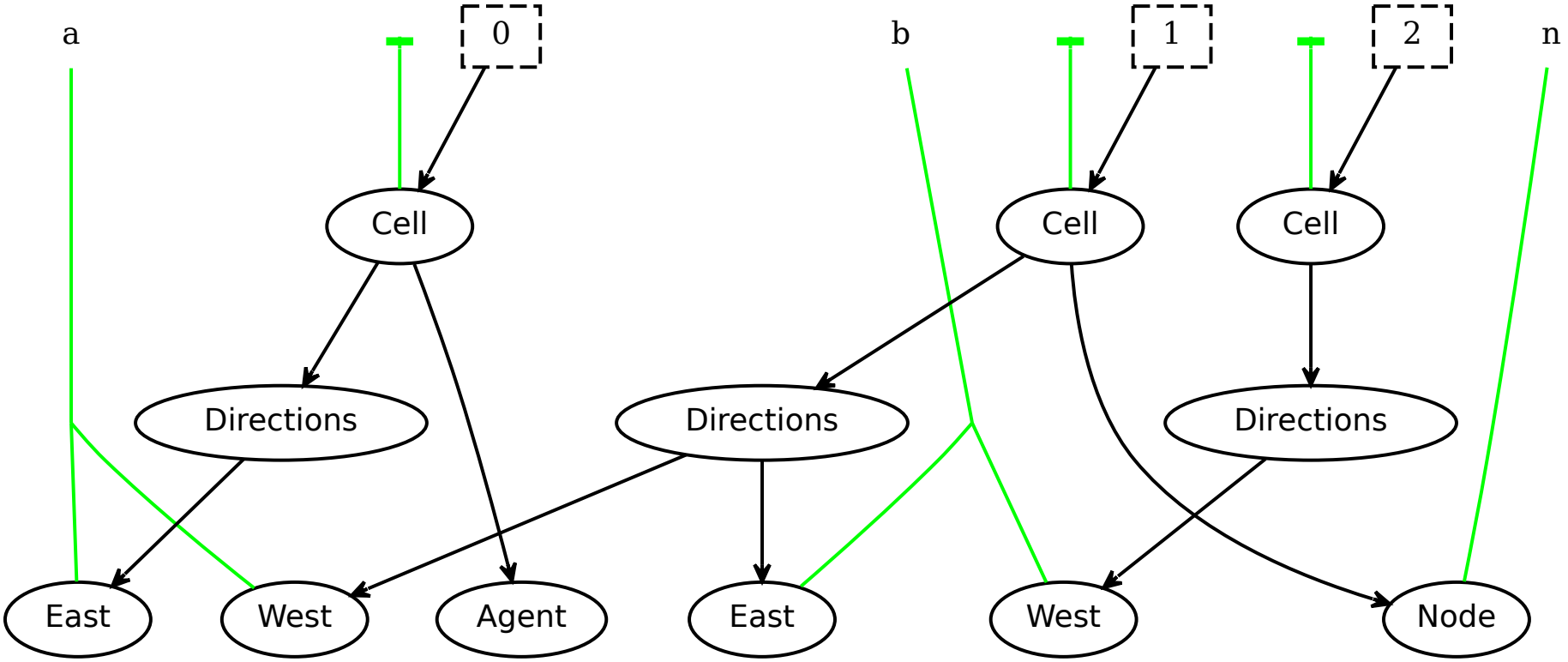
```
action goIn
  react goIn = Cell{x}.(Agent | Node{n}.(Cell{n}
                                          | id)
                        | id)
               -[1.0]->
               Cell{x}.(Node{n}.(Cell{n}.(Agent
                                          | id)
                                | id)
                        | id);
end
```

Entering/Leaving a Room

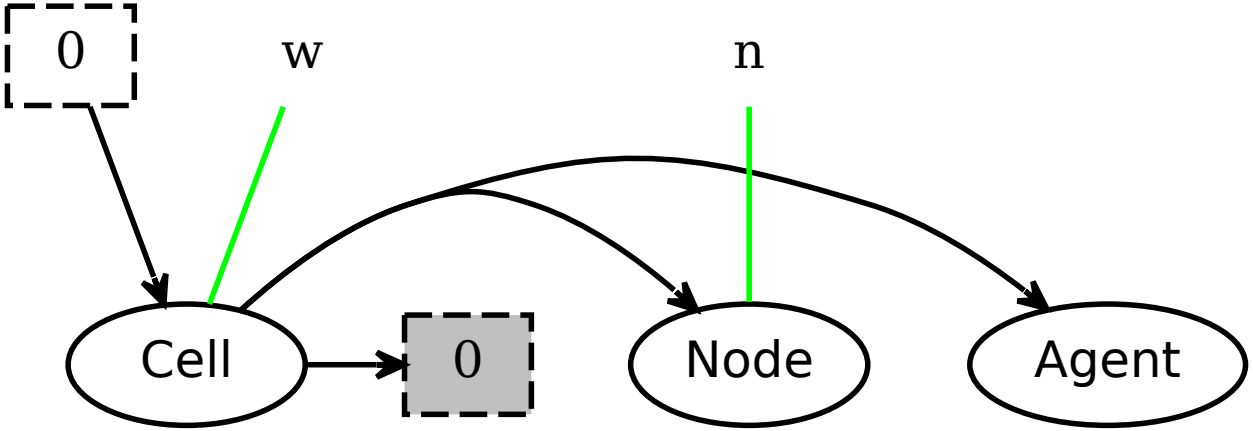
Action rewards

```
action goIn[1]
  react goIn = Cell{x}.(Agent | Node{n}.(Cell{n}
                                          | id)
                        | id)
               -[1.0]->
               Cell{x}.(Node{n}.(Cell{n}.(Agent
                                          | id)
                                 | id)
                        | id);
end
```

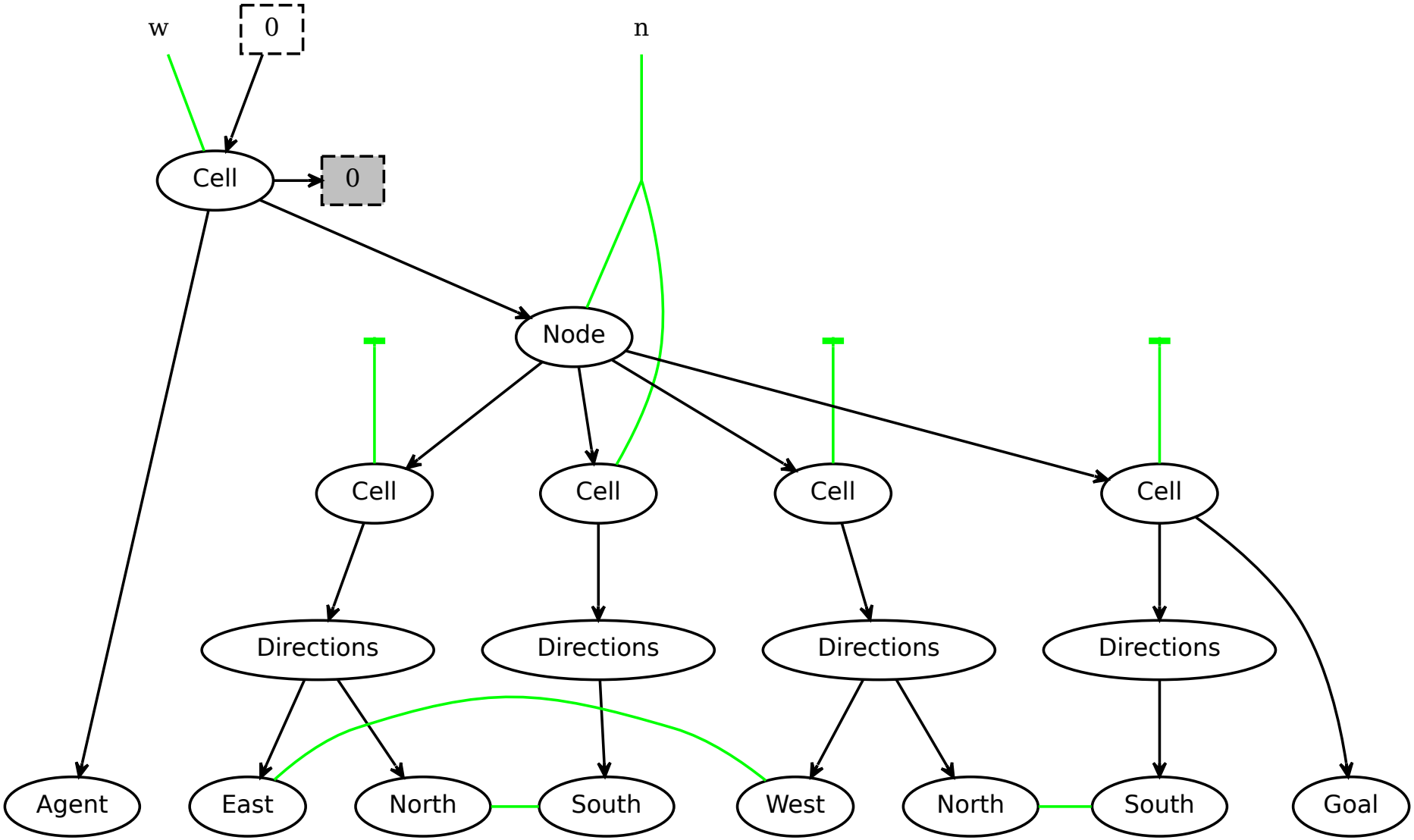
Initial State



Opening the Door



Opening the Door



Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

Tracking Time with State Rewards

- Blanket predicate

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

- Blanket predicate
- Nondeterministic Bigraphical Reactive System

Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

- Blanket predicate
- Nondeterministic Bigraphical Reactive System
- Initial state

Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

- Blanket predicate
- Nondeterministic Bigraphical Reactive System
- Initial state
- List of reaction rules (grouped into priority classes)

Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

- Blanket predicate
- Nondeterministic Bigraphical Reactive System
- Initial state
- List of reaction rules (grouped into priority classes)
- List of predicates

Tracking Time with State Rewards

```
big agent = Agent;

begin nbrs
  init initialState;
  rules = [ {...}, {...} ];
  preds = { agent[1] };
end
```

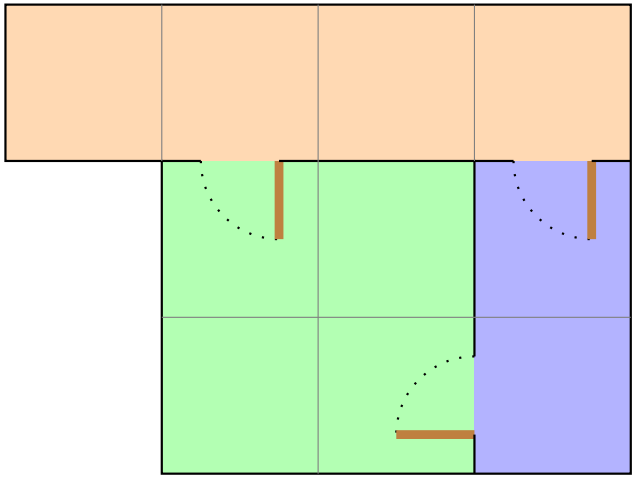
- Blanket predicate
- Nondeterministic Bigraphical Reactive System
- Initial state
- List of reaction rules (grouped into priority classes)
- List of predicates
- Predicate rewards (optional)

Extensions

- Multiple rooms (make each `Node` uniquely identifiable)

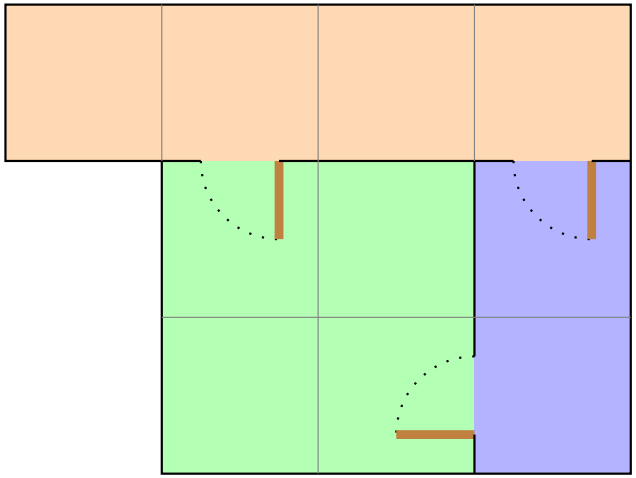
Extensions

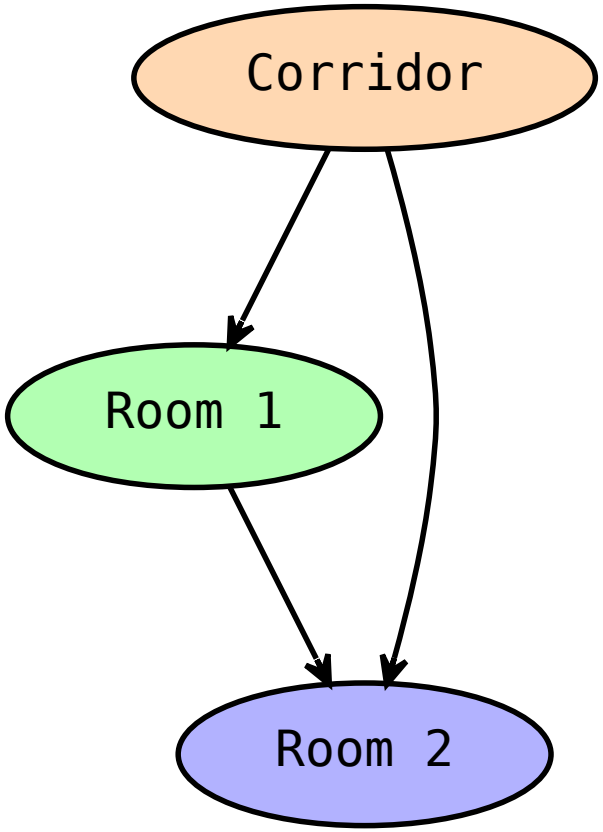
- Multiple rooms (make each `Node` uniquely identifiable)
- Arbitrarily complex configurations (via bigraphs with sharing)



Extensions

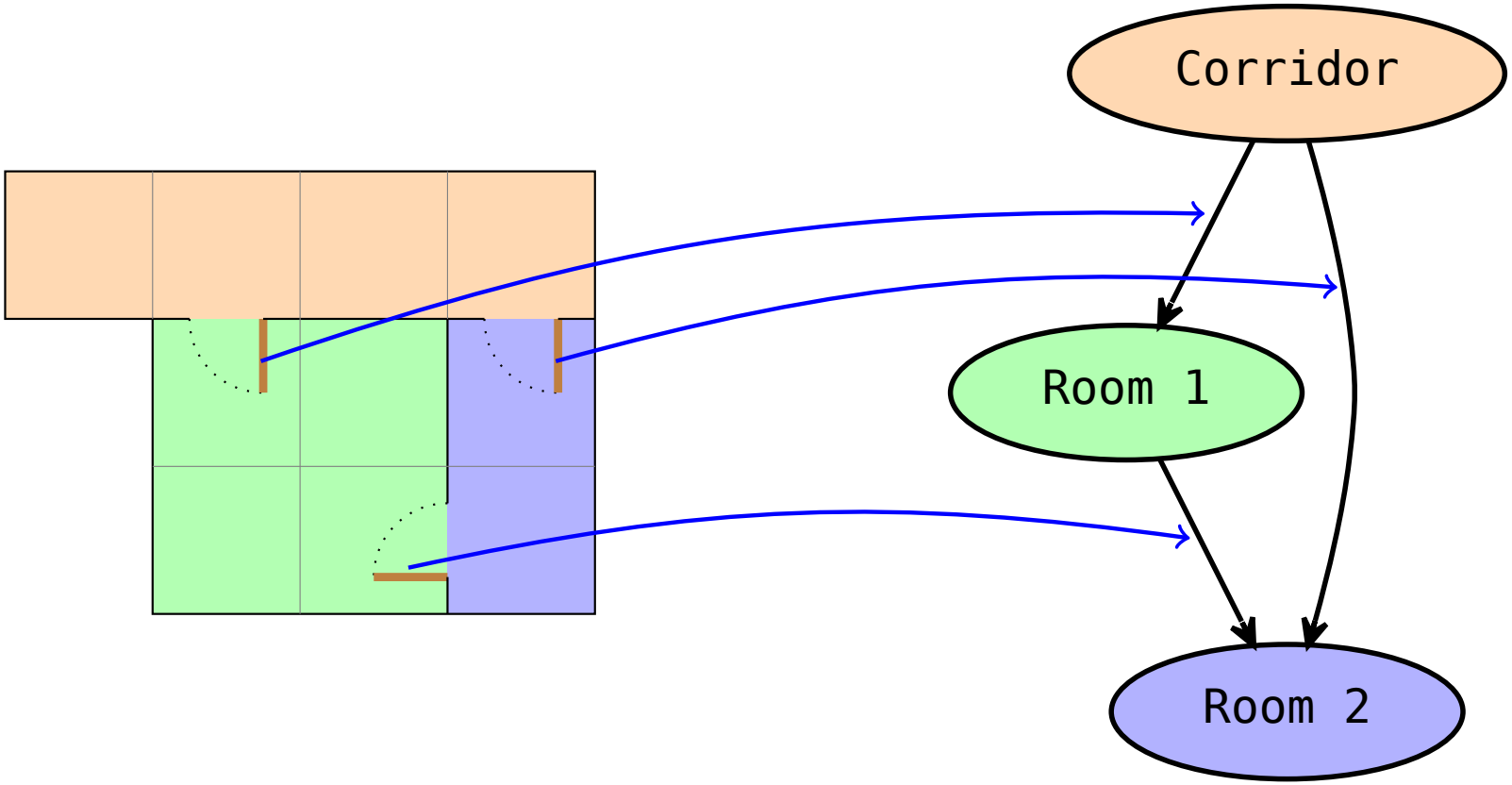
- Multiple rooms (make each `Node` uniquely identifiable)
- Arbitrarily complex configurations (via bigraphs with sharing)





Extensions

- Multiple rooms (make each `Node` uniquely identifiable)
- Arbitrarily complex configurations (via bigraphs with sharing)



Limitations of the Model

- One door per cell
  - Otherwise which door the agent uses would be random
  - Workaround: use more cells
- Two ideas in one: discovering space & entering an inner space

A New Interface

jupyter Example Last Checkpoint: a minute ago (autosaved) Logout

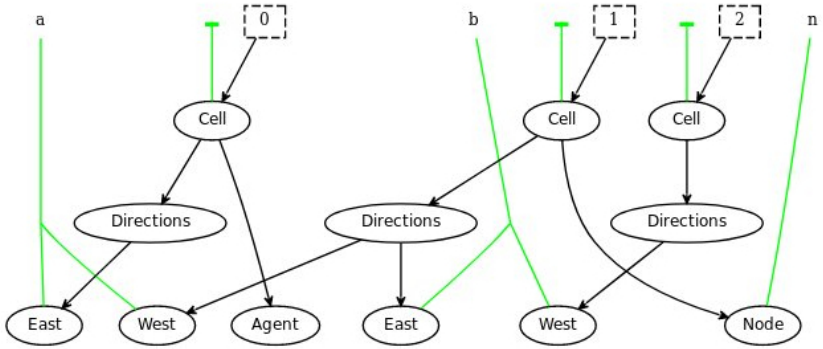
File Edit View Insert Cell Kernel Widgets Help Trusted BigraphER 1.7.0 (OCaml 4.06.0)

Run Code

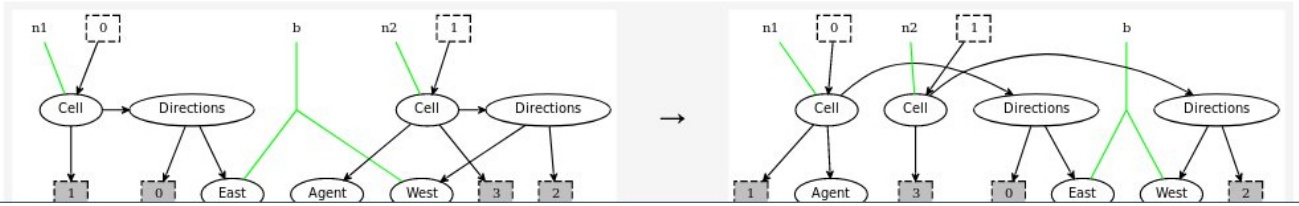
In [1]:

```
ctrl Cell = 1;
ctrl Directions = 0;
ctrl Node = 1;
atomic ctrl Agent = 0;
atomic ctrl East = 1;
atomic ctrl West = 1;
big initial = /x /y /z (
              Cell{x}.(Directions.East{a} | Agent)
            || Cell{y}.(Directions.(East{b} | West{a})
                     | Node{n}.1)
            || Cell{z}.Directions.West{b});
react goWest = Cell{n1}.(Directions.(East{b} | id) | id)
            || Cell{n2}.(Directions.(West{b} | id) | Agent | id)
            -[1.0]->
              Cell{n1}.(Directions.(East{b} | id) | Agent | id)
            || Cell{n2}.(Directions.(West{b} | id) | id);
```

Out[1]: initial



Out[1]: goWest



A New Interface

- Similar workflow to other Jupyter notebooks
- Syntax highlighting
- Visualisation of bigraphs and reaction rules
- Full and partial transition diagrams
  - with state bigraph preview on mouseover
- Backwards compatible to run OCaml code

Available at

`https://github.com/dilkas/bigrapher-jupyter`

Conclusions

+ Easy to represent complicated spatial structures and uncertainty about them

Conclusions

+ Easy to represent complicated spatial structures and uncertainty about them
+ A direct visual representation of the modelled situation

Conclusions

+ Easy to represent complicated spatial structures and uncertainty about them
+ A direct visual representation of the modelled situation
+ Succinct and easy to modify

Conclusions

+ Easy to represent complicated spatial structures and uncertainty about them
+ A direct visual representation of the modelled situation
+ Succinct and easy to modify
+ Can generate any MDP

Conclusions

- + Easy to represent complicated spatial structures and uncertainty about them
- + A direct visual representation of the modelled situation
- + Succinct and easy to modify
- + Can generate any MDP
- + Transition system can be exported to PRISM
  - – with some limitations (e.g., no transition rewards)

Conclusions

- + Easy to represent complicated spatial structures and uncertainty about them
- + A direct visual representation of the modelled situation
- + Succinct and easy to modify
- + Can generate any MDP
- + Transition system can be exported to PRISM
  - – with some limitations (e.g., no transition rewards)
- – Probabilities are constant and cannot easily depend on state

Conclusions

- \+ Easy to represent complicated spatial structures and uncertainty about them
- \+ A direct visual representation of the modelled situation
- \+ Succinct and easy to modify
- \+ Can generate any MDP
- \+ Transition system can be exported to PRISM
  - − with some limitations (e.g., no transition rewards)
- − Probabilities are constant and cannot easily depend on state
- − Some simple ideas are impossible or hard to implement
  - adding/subtracting integers
  - set membership check

Conclusions

+ Easy to represent complicated spatial structures and uncertainty about them
+ A direct visual representation of the modelled situation
+ Succinct and easy to modify
+ Can generate any MDP
+ Transition system can be exported to PRISM
  - with some limitations (e.g., no transition rewards)
- Probabilities are constant and cannot easily depend on state
- Some simple ideas are impossible or hard to implement
  - adding/subtracting integers
  - set membership check

Thank You!