MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Locomotion of self-organizing robots

Author: Lorena Eleonora Lusso

Advisor: Vera Sacristán Adinolfi

Department: Applied Mathematics

Academic year: 2012/2013

Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Locomotion of self-organizing robots

Lorena Eleonora Lusso

Master thesis
Supervisor: Vera Sacristán Adinolfi
Department of Applied Mathemathics

Master in Advanced Mathemathics and Mathematical Engineering
Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

October 7, 2013

Contents

Chapter 1

Introduction

1.1 Goal

We present here a collection of distributed algorithms for the locomotion of rectangular and histogram-shaped square-lattice-based modular robots, on free ground and in the presence of obstacles.

1.2 Context

Modular robots are systems composed of several different identical module types, that can be manually or self-reconfigured to form a different robot; due to their structure, these robots have the potential to be extremely versatile in the tasks they can perform, since they can reconfigure to adapt to different environments and tasks. Each module of the system can contain electronics, sensors, computer processors, memory, power supplies; they can also contain actuators that are used for manipulating their location in the environment and in relation with each other. In some cases they present two or more connectors, and they have the ability to automatically connect and disconnect themselves to and from each other.

In our work we focus our interest on self-reconfiguring modular systems: robots capable of utilizing its own system of control to change their overall structural shape, mostly consisting of a small number of fixed-shaped units which can move relative to their neighbours, rearranging the connectivity of their parts.

Two basic types of methods of articulation that self-reconfigurable mechanisms can be utilized by modular robots to reshape their structures: chain reconfiguration and lattice reconfiguration. Lattice architectures have their units connecting their docking interfaces at points into virtual cells of some regular grid, while chain structures do not use a virtual network of docking points for their units: the units are able to reach any point in the space; because of their configuration, chain structure are usually more versatile, but it is also more difficult to accomplish a reconfiguration step; moreover, such systems are computationally difficult to represent and analyze. Our work focuses on lattice-based structures. Modular robotic systems are also generally classified depending on the design of their modules; in our setting, modular robots are

considered homogeneus, as all the modules have the same shape and dimensions, in contrapposition to heterogeneous modular robot systems, which have different modules, each of which do specialized functions.

The reconfiguration methods are usually classified into deterministic and stochastic. The reconfiguration methods we present are completely deterministic, as the exact location of each unit is known at all times, while in stochastic methods units move around using statistical processes.

Many examples of self-reconfiguring robots have been phisically built, with a wide variety of action capabilities. We present here some examples that fit our setting, i.e. some example of lattice-based modular robots that have been built, in Figures 1.1, 1.2, 1.3 and 1.4.

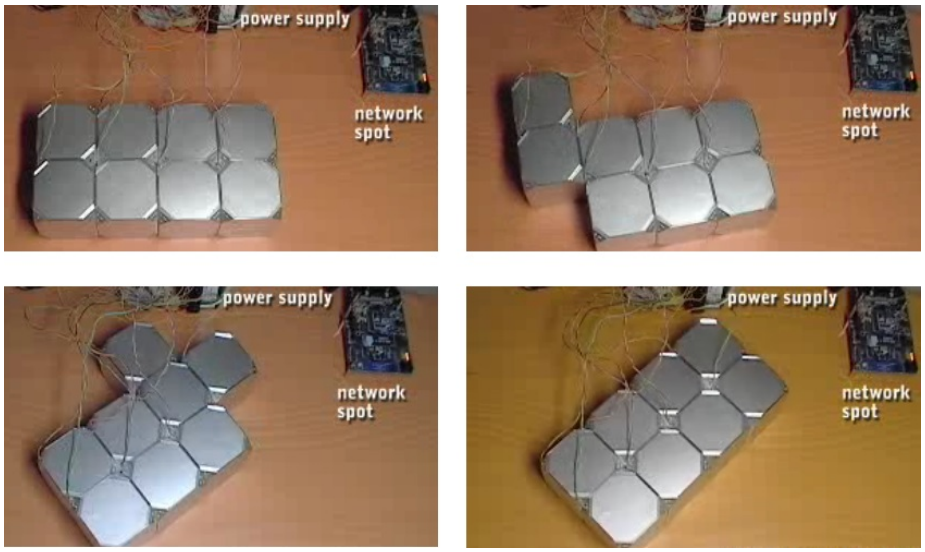


Figure 1.1: Eight modules of the EM-Cube [1]; two modules move on three different sides of the system, reconfiguring the initial shape at the end of the movement.

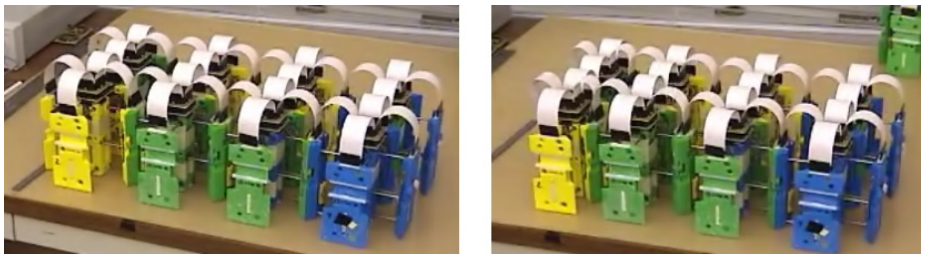


Figure 1.2: Eight modules of the Crystal system [2] performing locomotion.

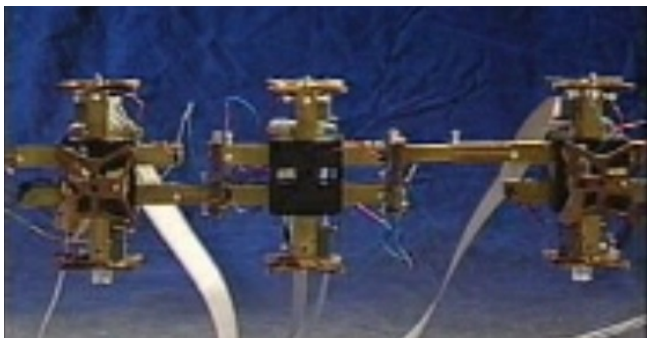


Figure 1.3: Three modules of the Telecube [5] joint togheter.

Figure 1.4: Six modules of the Micromodules [3] joint togheter.

1.3 Framework

In our framework, a robot is a connected set of identical modules located in a 2-dimensional square lattice. Each module of the robot is assumed to occupy one grid cell, and to be attached to its direct grid neighboring modules. The modules are assumed to perform some basic movements relative their neighbors, applying one of the two relative movements depicted in Figure 1.5.

In our framework we assume that each module attaches immediately to each one of its neighboring modules as it changes its position.

Each module disposes of an internal memory, in which an internal state (active, passive, stop...) is memorized in a short text string, and the value of some counters can be stored and used to perform some simple operations as comparisons, sums and subtractions.

In order to be able to apply a movement, each module needs to able to get

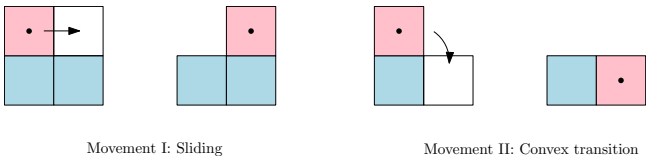


Figure 1.5: Advance moves: (a) slide and (b) convex transition.

some information not only about its own condition, but about some external grid cells too: in our framework, modules are able to understand if a given grid cell is empty or occupied, either by another module or by a physical obstacle, to receive information about the value of the counter of other modules, and to detect their internal states.

The communication with the rest of modules is limited by proximity restrictions; we assume that each module is able to get information about positions belonging to the first and to the second neighborhood of grid cells (refer to Figure 1.6); the possibility of questioning the grid cells of the second neighbor is restricted to the essential cases, as more complicated to achieve in practical situations; such cases are specified along the document.

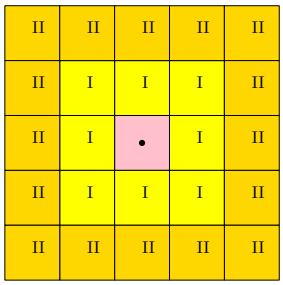


Figure 1.6: Illustration of the grid cell positions belonging to the first and the second neighbor of the current module; the current module is depicted in pink with a dot in its center, the grid cell of the first neighbor are yellow and indicated with a I while II stands for the cells of the second neighbor, this last one depicted in orange.

In each one of the algorithms presented in the following chapters, some information about the overall shape of the system (such as the height of a rectangular system, the number of columns for an histogram, the height of a column) is stored into each module; such information doesn't have to be communicated to the modules by an external agent, but it can be obtained by the modules alone in a separate previous phase, controlled by the rules for the bounding box presented in [6]. For this reason we don't discuss here the process of getting such information, and we assume that the modules have already analyzed the shape of the system before the application of our rules. Our algorithms consist in set of rules that all modules run at the same time, but modules can apply at most one rule at each evaluation round; this is achieved by giving to the rules a particular structure, that makes them pairwise incompatible, or in some way

ordered by priority. Each rule, in fact, has the following structure: a priority, a precondition, and a postcondition. Priorities are represented as integers, and assigned to different rules a order of importance: any module whose situation satisfies at the same time the precondition of two or more different rule, will apply the rule with maximal precondition among them.

A precondition is a set of conditions that need to be satisfied by the module and its neighbors (first and second neighborhood); in particular, each module need to check its own internal state (defined a text string indicating if the module is active, inactive, an obstacle...), the state of its neighboring grid cells, the value of its internal counters and of the counter of its neighbors, to compare its own counters to the ones of other neighbors, and to check if the neighboring grid cells are either empty or occupied by other modules.

A postcondition is a combination of one or more of the following actions: a change of the internal state of the module, a movement in a given direction with a change of the attachments with of the module, and a change of the internal counters of the module.

For each algorithm presented we illustrate the strategy of the movement, a proof of correctness of the rules and study of the complexity of each set of rules. Moreover, each set of rules is implemented in a simulator; such simulator applies the set of rules memorized over a set of modules defined by the user; as we said, the simulator run the entire set of rules to each module at the same time. At each evaluation round, each module applies all the rules that are applicable to its situation, depending on the preconditions and the priority of the rule; such application is performed independently from the rest of modules.

Together with the presentation of the strategy of the movement then, the detailed set of rules and visual examples of the implementation of such rules are presented.

The detailed functioning of the simulator can be read in [7].

1.4 Related work

Our work is mostly inspired on [8], which presents some sets of rules dedicated to the free locomotion of rectangular systems and to the overpassing of histogram-shaped obstacles. In [8] the authors use three different evaluation models which happen to be essentially sequential; in the first one, the modules are evaluated in a set cyclical ordered, with no variation in the relative delay between any two cells. In the second execution model, a different permutation is used at each round; and in the third model, at each turn one randomly chosen module applies the rules and turns disappear. In each of the three cases, the rules are applied to each cell one by one in sequence, not all togheter as a set. Some practical experiments that we have performed in the simulator [7] prove that their essentialy sequential rules, implemented in a parallel setting, generate disconnections and collisions. Our goal has been to design alternative algorithms in order to effectively parallelize the execution of the rules by all the modules; our modules execute then the rules in a synchronous parallel way.

1.5 Structure of the document

The document is ideally divided into three parts; the first part, formed by the first five chapters, treats the in-shape locomotion of systems initially configured as rectangles; in particular we present the following algorithms: rules for the free locomotion of rectangular systems (Chapter I), locomotion of a rectangle in the presence of inferior histogram obstacles (low obstacles in Chapter II and high obstacles in Chapter III), and locomotion of a rectangle in the presence of superior histogram and general-shaped obstacles (Chapter IV). Chapter V is devoted to the tunneling of a rectangle, i.e. the locomotion of a rectangular system in the presence of tunnels, formed by inferior and superior histogram shaped obstacles.

The second part, from Chapter VI to Chapter VIII is devoted to the study of the in-shape locomotion of histogram-shaped systems; in Chapter VI we treat the free locomotion of a general histogram, in Chapter VII the locomotion of an histogram in the presence of low and high histogram-shaped inferior obstacles, and in Chapter VIII the locomotion under superior histogram obstacles.

The third part, formed by Chapter IV, contains an introduction to the problem of the overpassing of general shaped obstacles. The document ends with some conclusions, a discussion about further work, and the references mentioned along the text.

Chapter 2

Rectangle locomotion

2.1 Goal

The purpose of the set of rules presented in this chapter is to produce the eastward locomotion of any modular robotic system initially configured as a connected rectangle without holes. The locomotion is performed on a free plane ground (a horizontal line) without obstacles. The rules are divided into two sets: the first one produces the proper locomotion, while the second one reconfigures the system into the original shape at the end of the locomotion.

2.2 Strategy

The rules produce the locomotion by making each module in turn move from the back of the group, over the top, and locate on the front of the group to form a new column. The first move is performed by the topmost module of the leftmost column; the other modules of the column follow it until the second column becomes the leftmost and is free to move in the same way; see Figure 2.1 for an illustration.

Along the locomotion the modules pass through two different states: *inactive* and *active*; all the modules are initially inactive, they progressively activate and move relatively to the inactive modules while applying the rules, and deactivate again as they settle on the front of the system. These changes of state run until the system reaches its final position, when modules gradually change their state to *stop* through the stop rules, and reconfigure again into a rectangle.

As the first module sets its state to *stop* no change-position rule is applicable and the configuration is blocked. The modules stay connected to the others during the entire locomotion, so that the system always consists in a connected set while moving.

2.3 Advance rules

For the locomotion the modules use five advance moves: North, North-east, East, South-east and South.
Each of these rules, if applicable, has as effect the change of a module position

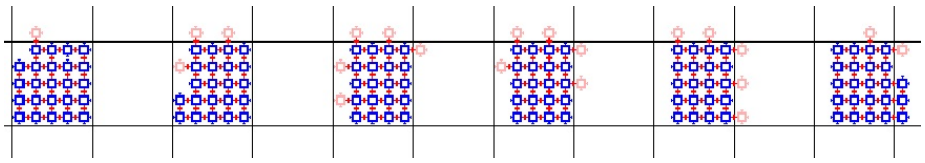


Figure 2.1: Execution of the action rules: the leftmost column walks on the top of the rectangle and settles on the front of the system. Active modules are depicted in pink, inactive modules are depicted in blue.

in a different direction, as illustrated in Figure 2.2.
As we can notice, the goal position is always required to be empty, and modules are allowed to move only on inactive modules.

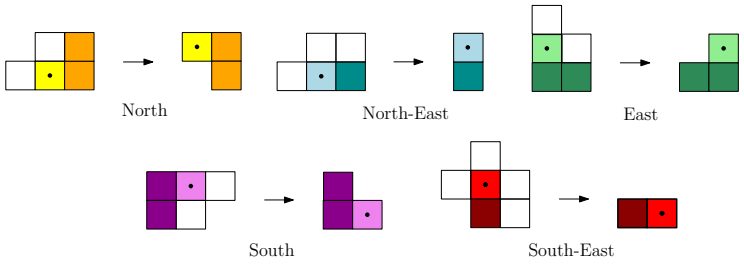
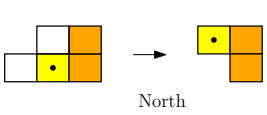


Figure 2.2: Advance rules: change position. In this graphical representation the left picture of each rule represents the preconditions that need to be satisfied in order to perform the movement while, on the right, the result of the application of the rule is depicted. The darkest modules are inactive, and the empty squares correspond to empty positions of the grid; the module applying the rule is depicted in a lighter color, and has a dot.

We can see in details the structure of these rules by taking as an example the North rule, illustrated in Figure 2.3:



Move North
2
State≠Stop & (-1,0)=(0,1)=empty & State(1,0)=State(1,1)=Inactive
Move (0,1) & State=Active

Figure 2.3: Detailed example of the structure of one of the advance rules.

Every action rule, as the North rule, has priority equal to 2, and is applicable only to modules that are either inactive or already active.

When the preconditions are satisfied the rule is applied, and the state either changes from inactive to active, or stays active if the module had already been activated. When none of these rules is applicable to a module, a change of state is obtained through the deactivation rule depicted in Figure 2.4.

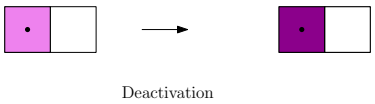


Figure 2.4: Advance rules: deactivation.

The priority of the deactivation rule is 1, so that moving has always priority with respect to the deactivation, and a module deactivates only when it cannot move in any direction.

The action rules presented in Figure 2.2 need to be modified in the case of a rectangular system formed by only one column of modules. In this case, to produce the locomotion of the system a modification to the South-east rule is necessary in order to avoid disconnections. The second version of the South-east rule is depicted in Figure 2.5.

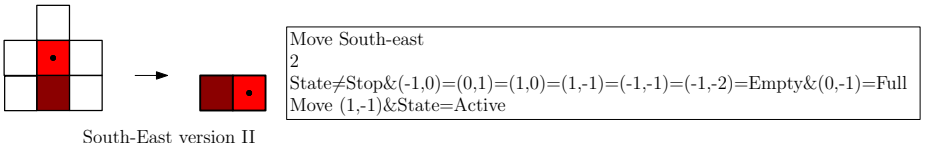


Figure 2.5: Advance rules: second version of the South-east rule for the case of one column.

2.4 Stop rules

As we don't want the locomotion procedure to be open loop some stop rules are needed in order to break the movement. During the action phase, the first module on the top of the first column starts the locomotion by moving North-east, and allows the other modules of the same column to change their state to active and apply the other action rules. By stopping such a module we can therefore stop the movement of the first column, and as a result the movement of all the system. In fact, when the highest module on the left column is blocked, as soon as the active modules have completed the rightmost column no other rule different from the stop rules is applicable. In order to make use of these facts, before starting the locomotion we fix a value and store it in the counter C01, which is memorized into each module of the system. This value will control the number of rounds the modules will move. Along the locomotion, each time a module moves North-east a second counter C00 is increased by 1; as long as this second counter is smaller than the fixed one C01, the action continues. As some module reaches the desired value of C00 and happens to be in the correct position, the stop rule may be applied, as we can see in Figure 2.6. Notice that this rule has higher priority than any of the priorly described ones. This guarantees that it will be applied whenever the preconditions are fulfilled.

As the first module changes its state to *stop*, the others can detect its new condition and change their own state to *stop*; this is done through four stop rules that are represented in Figure 2.7. No other rule is applicable to a module which

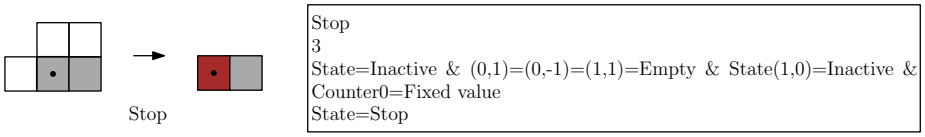


Figure 2.6: Stop rule for the first module.

is set in the stop state, so the locomotion ends, and the system is reconfigured into a rectangle as soon as the last active modules change their state to inactive, once settled in the last column, and then stop.

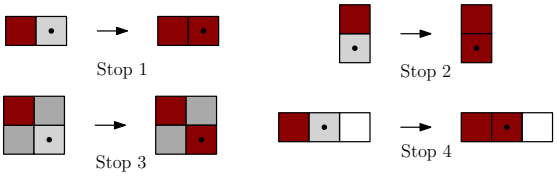


Figure 2.7: Stop rules for the rest of the modules: any inactive module meeting the preconditions and with a neighbor in the stop phase (dark red in the picture) changes its own state from *inactive* to *stop*. Dark gray cells are occupied cells.

2.5 Correctness

In this section we prove that the rules presented in this chapter serve our purposes, that is to make a rectangular robotic system advance from east to west over a flat ground without obstacles, and reconfigure after a given number of rounds. In particular, we prove the following:

1. No collisions are produced during the movement.
2. The system stays connected during the whole movement.
3. There is always at least one rule that is applicable to some module of the system.
4. The rules actually produce a locomotion of the system from west to east.
5. The stop rules block the locomotion and reconfigure the system into a rectangle.

Lemma 1 *Each module can apply at most one action rule at a time.*

Proof: The result is easily seen by a comparison between the preconditions of the five action rules, which turn out to be pairwise incompatible. Without loss of generality, we can examine the rules North and North-east: notice that one of the requirements for the application of North is that the cell grid with relative coordinates $(1, 1)$ has to be occupied by an inactive module, while North-east

requires the same cell to be empty in order perform the change of position. This incompatibility stands for any pair of these rules, so each module either can apply one action rule, or can't apply any of them. □

Proposition 1 *No collisions are produced during the movement.*

Proof: We want to prove that two different modules cannot satisfy the preconditions for moving to the same grid cell at the same time. Due to Lemma 1 we only need to worry about conflicts created by the application of one rule at a time, and not by changes of position resulting from combinations of different movements. The case of conflicts between the North and the South rule is easily excluded: any module able to apply North is a module attached to the left side of the system, while any module applying South is attached to the right, and the two movements cannot conflict. The only other type of conflict which could occur is the one that would be generated by rules with a common direction such as North-east and North, North-east and East, South-east and East, South-east and South or North-east and South-East, but we can easily discard these possibilities too: without loss of generality we can examine the case of North-east and North. Let's suppose that A is a module that intends to occupy the grid cell with relative coordinates $(0, 1)$ by moving North. A precondition for the movement is that the cell grid in relative position $(-1, 0)$ is empty. But any module B willing to occupy the same grid cell by applying North-east would occupy exactly such position, and this is a contradiction (see Figure 2.8). The rest of the possible mentioned conflicts present the same contradictions, and therefore can be excluded. □

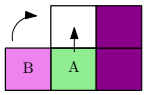


Figure 2.8: The module A cannot move North, as B is occupying the cell grid $(-1, 0)$.

Lemma 2 *Any module able to apply either the East or the South rule is an active module, as it has already been activated by the application of previous rules.*

Proof: Neither the East nor the South rule can be the first rule applied by a module as the initial configuration of the system is a rectangle, and the preconditions of both rules clearly show that a module which is able to apply one of these two rules cannot be part of a rectangular system. The only possibility left is that a module has been deactivated by the deactivation rule, and restarts the movement applying East or South; we will see that this is impossible too. As the priority of the deactivation rule is smaller than the priority of any other rule, modules only deactivate when no action rule is applicable. As Lemma 1 guarantees that no collisions occur during the locomotion, any active module is free to move until it reaches the rightmost column on the front and settles on top of another deactivated module. In this case it will activate again only when it will be able to apply the North rule again, so this last possibility can be discarded too. □

Proposition 2 *The system stays connected during the whole movement, i.e. no rule produces a disconnection between the module applying it and the system.*

Proof: Due to Lemma 1, the only cases to study are the ones originated by the application of one rule at a time. All the post-conditions of the action rules guarantee that the current module is connected to one of its neighbors after the movement, we will call it the 'support module' of the rule; the only situation in which a disconnection could occur is the one in which while a module applies a rule and intends to connect to the support module, this last one applies an action rule itself, leaving its grid cell empty. The only way to exclude this possibility is to analyze the possible support modules, and prove that there is no rule that they could be applying. We can notice that the support module is always supposed to be inactive, therefore Lemma 2 guarantees that it cannot be applying neither the East nor the South rule. Inactive modules able to apply the South-east rule can only be found in the particular case in which the initial configuration consists in a rectangle formed by only one column. In this case, the topmost module would activate directly by applying the South-east rule, while the other modules cannot apply any rule; so this module cannot play the role of support module for any other, and this situation cannot generate disconnections. Inactive modules able to apply North and North-east rules can only be found in the leftmost column, in the case of a rectangle formed by more than one column. The only inactive module able to apply the North-east rule is the module which starts the movement of its own column; we can easily see that this module cannot be the support module of any rule, as no one of its neighbors satisfy the preconditions of any of the rules, as illustrated in Figure 2.9.a. The inactive modules able to apply North are the modules which follow the movement of the first one of their column, activating by moving North. As in the last case, no one of their neighbors is able to move (see Figure 2.9.b), so we can discard this last case too. □

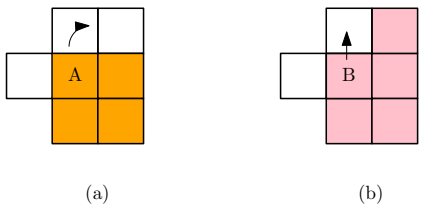


Figure 2.9: (a) The inactive module A applying the North-east rule cannot be the support module for any of its neighbors. (b) The inactive module B moves North, and no other neighboring module can move.

Proposition 3 *There is always at least one rule that is applicable to some module of the system, so the system cannot get stuck during the movement.*

Proof: We can analyze the movement of the leftmost column and proceed by induction. From the initial configuration of a rectangle, there is a one and unique module able to apply a rule: the first module on the top of the leftmost column; as it applies the North-east rule, it leaves space to the other modules of its column to activate and move North and North-East progressively. The

first module, followed by the others of its column, moves East on the top of the system, South-East, and South along the rightmost column. As these modules make their way to the end of the rectangle, they form a new rightmost column; the second leftmost column becomes the first, its topmost module is free to move and the motion is repeated. As there is always a leftmost column which is at some moment free to restart the movement, the system can always move further and there is always some rule applicable. □

Proposition 4 *The rules produce a locomotion of the system from west to east.*

Proof: As there is always one rule applicable to the system because of Lemma 3, and each one of the rules North-east, East and South-east has a component in the east direction, the only thing we need to prove is that the rules don't produce an alternation of opposite movements without moving the system in any direction. The only two rules that don't produce a movement in the east direction are the North and the South rules, but we can easily notice that any module which performs a movement in the North direction cannot produce one in the South direction until it has reached the opposite side of the rectangular system; this locomotion of the modules from the back to the front produces an overall movement towards east without any risk of oscillations. □

We have proved the following:

Theorem 1 *The rules described in section 2.3 allow any rectangular configuration of modules to advance eastwards while keeping the system connected.*

A last comment is to be given about the stop rules described in section 2.4. Given the strategy of the locomotion movement, it is straightforward that by stopping the topmost module of the leftmost column the locomotion is blocked. As soon as the remaining active modules have moved from the top of the rectangle to the right and have formed the new rightmost column, the system is reconfigured into a rectangle. The following result is then immediate:

Proposition 5 *The rules proposed in section 2.4 produce the interruption of the locomotion and the reconfiguration of the system into a rectangle.*

2.6 Complexity

Neighbourhood During the locomotion each module is only able to check if any grid cell sharing either an edge or a vertex with it is empty or occupied by another module; in this last case, it can obtain information about the state of such module. No information about any other position is needed.

Memory and computation Locomotion only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to memorize a fixed value in a counter, and to check if preconditions of rules are fulfilled.

Number of moves For an advance length of k rounds, and a rectangular system of dimension $h \times b$, each module performs $O(k(h+b))$ changes of position, $O(k)$ deactivations (it deactivates once at each round) and stops once, therefore each module performs at most $O(kn)$ rules, where n is the

number of modules of the system; the number of rounds k depends linearly on the distance covered by the system during the locomotion.

Communication A constant size communication is performed during the application of each rule, so the communication performed by each module is $O(n)$.

Number of time steps In a synchronized execution, for an advance length of k rounds and a rectangular system of dimension $h \times b$, locomotion is performed in $O(k(2hb))$ steps; this can be seen by charging to each column the steps starting from the activation of the topmost module of the column to the first movement of the topmost module of the following column, and multiplying by the number of rounds. Notice that the number of rounds k is directly proportional to the distance between the starting and the goal position. As reconfiguration too only requires $O(n)$ steps, the overall number of time steps is $O(kn)$.

2.7 Rules

The algorithm is based on 11 different rules; before the locomotion the number of rounds needs to be stored in counter C01.

Rule name: North Priority: 2 Preconditions: !SSTOPP N001* T1,1,INACT Postconditions: P0,1 SACTIV A****

- **Rule**: North
- **Priority**: 2 (the priority is the same for each action rule).
- **Preconditions**:
!SSTOPP: the rule is not applicable to modules set in the *stop* state.
N001*: concerns only the modules of the leftmost column, without a North neighbor.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A****: the module attaches afterwards if was attached before, when possible.

Rule name: North-east Priority: 2 Preconditions: !SSTOPP N001* T1,0,INACT E1,1 Postconditions: P1,1 SACTIV A***1 C000 + C000 0001

- **Rule**: North-east
- **Priority**: 2 (the priority is the same for each action rule).
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N001*: concerns only the modules of the leftmost column, without a North neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: East Priority: 2 Preconditions: !SSTOPP N0*01 T1,-1,INACT Postconditions: P1,0 A***1

- **Rule**: East
- **Priority**: 2 (the priority is the same for each action rule).

- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N0*01: the rule concerns only the modules without north nor west neighbors, but with a south neighbor; the presence of a module on the east is not important.
T1,-1,INACT: the support module needs to exist and to be set on *inactive*.
- **Postconditions**:
P1,0: the module moves east.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South-east Priority: 2 Preconditions: !SSTOPP N0001 E1,-1 Postconditions: P1,-1 SACTIV A*1**

- **Rule**: South-east
- **Priority**: 2 (the priority is the same for each action rule).
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N0001: concerns the modules with a south attachment and no other neighbor.
E1,-1: the goal position needs to be empty.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1**: the module attaches to its new west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South Priority: 2 Preconditions: !SSTOPP N*100 T-1,-1,INACT Postconditions: P0,-1 A***1

- **Rule**: South
- **Priority**: 2 (the priority is the same for each action rule).
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N*100: concerns the modules with a west neighbor, without a module on the east and on the south. The presence of a north neighbor is not important.
T-1,-1,INACT: the support module needs to exist and to be set on *inactive*.
- **Postconditions**:
P0,-1: the module moves south.
A***1: the module attaches to its new south neighbor, if present; if it was attached before and where still possible, it attaches to the other neighbors.

Rule name: Deactivation
Priority: 1
Preconditions: !SSTOPP N**0*
Postconditions: SINACT

- **Rule**: Deactivation
- **Priority**: 1 (the priority is smaller than the priority of any action rule; modules deactivate only if no action rule is applicable).
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N**0*: concerns the modules without a module on the east. The presence of a north, west or south neighbor is not important.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Stop
Priority: 3
Preconditions: SINACT N001* T1,0,INACT E1,1 = C000 C001
Postconditions: SSTOPP

- **Rule**: Stop
- **Priority**: 3 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N001*: concerns the modules with a east neighbor and without a module on the north and on the west. The presence of a south neighbor is not important.
T1,0,INACT: the east neighbor needs to be an inactive module.
E1,1: the grid cell of relative coordinates 1,1 needs to be empty.
= C000 C001: the value of the counter C000 needs to meet the value of the counter C001, this last one fixed before starting the locomotion. This last precondition controls the number of rounds.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop 1
Priority: 3
Preconditions: SINACT N*1** T-1,0,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 1
- **Priority**: 3 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N*1**: the module needs to have a east neighbor T-1,0,STOPP: the east neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 2
Priority: 3
Preconditions: SINACT N1*** T0,1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 2
- **Priority**: 3 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N1***: the module needs to have a north neighbor T0,1,STOPP: the north neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 3
Priority: 3
Preconditions: SINACT N11** T-1,1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 3
- **Priority**: 3 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
N11**: the module needs to have a north neighbor and a west neighbor.
T-1,1,STOPP: the grid cell with relative coordinates $(-1, 1)$ needs to be occupied by a module set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 4
Priority: 3
Preconditions: SINACT N*10* T-1,0,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 4
- **Priority**: 3 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
N*10*: the module needs to have a west neighbor and no module on the east.
T-1,0,STOPP: the west neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: South-east version II
Priority: 2
Preconditions:!SSTOPP N0001 E1,-1 E-1,-1
Postconditions: P1,-1 SACTIV A*1**

- **Rule**: South-east version II: case of one column.
- **Priority**: 2
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
N0001: concerns the modules with a south attachment and no other neighbor.
E1,-1: the goal position needs to be empty.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty; notice that this is the only modification needed to treat the case of one column.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1**: the module attaches to its new west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Chapter 3

Rectangle locomotion over low obstacles

3.1 Goal

In this chapter we discuss the eastward locomotion of a rectangular system in the presence of obstacles. In our setting, obstacles lay on the ground (a horizontal line) and are configured as histograms; the maximum height of the obstacles is one unit less than the height of the rectangle in the initial configuration. We refer to this kind of obstacles as 'low obstacles'. As in the free locomotion case, the rules are divided into locomotion rules and reconfiguration rules. As the reconfiguration is performed after the obstacles are traversed, it is performed exactly as illustrated in Chapter 2, and we will not discuss it another time.

3.2 Strategy

The strategy is similar to the one of the free locomotion, but with some adjustments that make it possible to crawl over the obstacles. Modules always move from the back of the group, over the top, and down on the front of the group. Before reaching an obstacle, the rectangle moves under the original locomotion rules described in Chapter 2; as the rectangle encounters an obstacle, the system fills the free spaces over the obstacle and flows over it until it reaches the ground another time. A third state is introduced in order to make the movement more

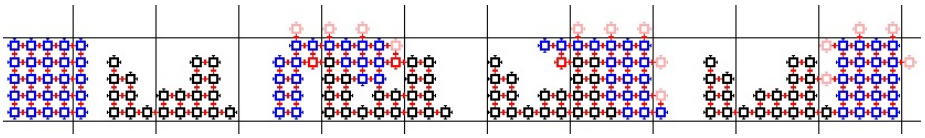


Figure 3.1: Crawling over a low obstacle: as the obstacle is reached, the modules move on the top of it through the use of bridges. Active modules are depicted in pink, inactive modules are blue, obstacles are black and bridges are the red modules.

fluid and faster, so modules pass now through three different states: *inactive*, *active* and *bridge*; this new state allows the system to avoid filling the bottlenecks of width 1 formed between the obstacle and the system itself, as we can observe in Figure 3.1. Each module set on *"bridge"* allows the other modules to pass by without having to fill all the spaces over the obstacles, and it does not move until all of them have passed.

During the locomotion, columns start the movement from their topmost module, and the other modules progressively activate by the a North rule until the last one is active and able to follow the others. This process is reversed for the column immediately on the left of a bridge as we can observe in Figure 3.2. During the whole movement the system maintains its height constantly equal to the initial one.

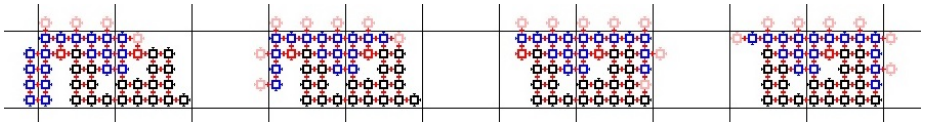


Figure 3.2: The order of activation of the modules of the column is reversed for the one preceding the bridge, and for the bridge itself: as the other columns have made their way and overpassed the obstacles, and no other inactive modules are left on the left of the obstacles, the strategy has to be inverted in order to avoid disconnections; the first module to move is now the bottommost, which activates moving North-west, and is followed by the others; as the possibility of disconnection is avoided the movement starts again from the topmost module.

3.3 Action rules

To perform locomotion in the presence of low obstacles, modules apply a set of nine action rules, depicted in Figure 3.3. Each of the action rules, if applicable, changes the position of a module in a different direction.

All the action rules, except for the South-East II rule, have priority equal to 3, as the North rule illustrated in Figure 3.3; the South-East II rule has priority 2, in order to prevent situations in which the same module is able to apply both South East and South-East II.

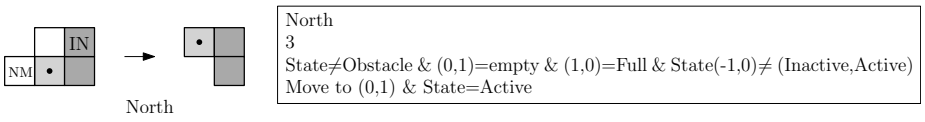


Figure 3.3: Detailed example of the structure of one of the advance rules.

3.4 Deactivation and bridges

Along with the action rules, two other rules are used in order to achieve locomotion over low obstacles: a rule for the creation of bridges and a deactivation

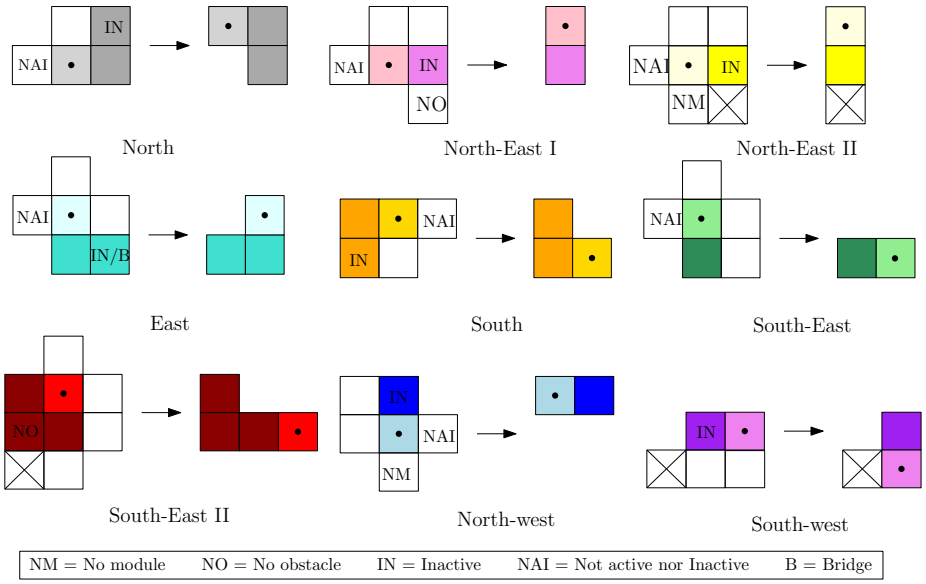
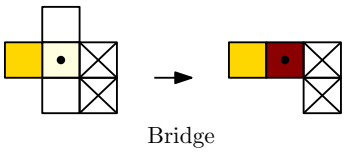


Figure 3.4: Action rules for the locomotion over low obstacles. In this graphical representation a colored square always indicates cells of the grid which need to be occupied; empty squares without any legend correspond to positions which need to be empty; the module applying the rule is the one marked with a dot and the crossed cells indicate cells occupied by obstacles. Legends denote some additional conditions that are required for the movement.

rule; both of these rules have the effect of changing the state of the module, without changing its position. Figure 3.5 illustrates the rule for the formation of a bridge.



Bridge

Bridge
6
State=Active & (0,1)=(0,-1)=empty & State(1,0)=State(1,-1)=Obstacle & (-1,-1)=Full
State=Bridge

Figure 3.5: Rule for the creation of bridges. The additional condition "State$(1,-1)$ =Obstacle" prevents the formation of bridges by modules lying on the floor.

Every time the first module encounters a obstacle, it changes its state into *'bridge'* and stays still until the other modules have moved over the top of it;

no action rule is applicable to a bridge apart from the North-East rule. As soon as all the modules have overpassed it, the bridge can apply the North-East rule and changes its state to active again.

The Bridge rule has priority greater than the priority of any other rule, so a module meeting the preconditions always creates a bridge.

The Deactivation rule, conversely, has priority smaller than any other rule, so a module deactivates only when it cannot apply any other rule. We can see the detailed Deactivation rule in Figure 3.6.

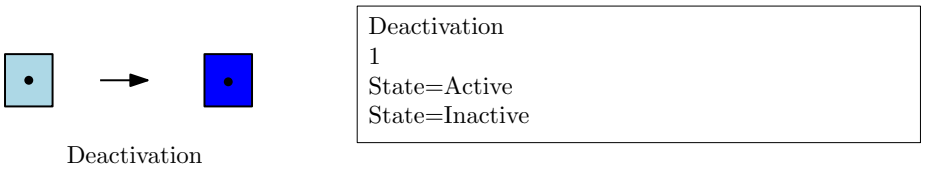


Figure 3.6: Deactivation rule

As in the case of free locomotion, if the system is formed by only one column of modules the rules presented need some modifications. In this particular case the use of bridges is not useful, as the system needs to attach to the obstacle and climb on it in order to maintain its stability and avoid disconnections. Therefore in this circumstance we eliminate the Bridge rule, and apply some further modification to the North-West rule, while all the other rules remain unvaried; we can observe the new North-West rule in Figure 3.7.

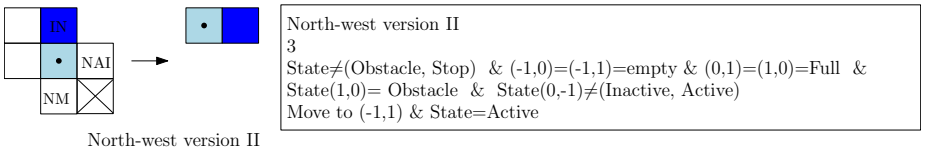


Figure 3.7: Advance rules: second version of the North-West rule for the case of one column.

3.5 Correctness

In this section we prove that the described rules actually produce an eastward locomotion of a rectangular robotic system in the presence of low obstacles. As in the case of free locomotion, we prove that:

1. No collisions are produced during the movement.

2. The system stays connected during the whole movement.

3. There is always at least one rule that is applicable to some module of the system.

4. The rules actually produce a locomotion of the system from West to East, on free ground and over low obstacles.

In order to prove these results, we firstly state and prove some preliminary results:

Lemma 3 *Each module can apply at most one action rule at a time.*

Proof: As the priorities of the South-East II rule and the Deactivation rule are smaller than the priorities of all the other rules, they don't need to be taken into account in this proof, as they cannot generate any problem. As all the other action rules have equal priorities, we need to check that they are pairwise incompatible, i.e. that a module cannot satisfy the preconditions of two of them at the same time. This is easily seen by analyzing the preconditions depicted in Figure 3.3.

Without loss of generality, we can examine the North and the North-East I rule: a requirement for the application of North is that the cell grid with relative coordinates (0,1) is occupied by an inactive module, while North-East I requires the same cell to be empty in order to change the position, so no module can satisfy both sets of preconditions at the same time.

We find a similar contradiction by analyzing each pair of these rules, so the rules are pairwise incompatible and modules can apply only one rule at a time. □

Proposition 6 *No collisions are produced during the movement.*

Proof: We want to prove that two different modules never move to the same grid cell at the same time while applying the rules. Due to Lemma 3 we only need to worry about conflicts created by the application of one action rule at a time, and not by changes of position resulting from combinations of different movements. The case of conflicts between the North and the South rule is easily excluded, as in the case of the free locomotion: any module able to apply North is a module attached to the left side of the system, while any module applying South is attached to the right, and the two movements cannot conflict. The case of conflicts generated by rules which produce the same change of position as North-East I and North-East II, or as South-East I and South-East II, is clearly impossible: two modules willing to move to the same grid cell performing the same movement need to occupy the same position in the first place, and this is not allowed in our configuration.

The only other type of conflict which could occur is the one that would be generated by rules with a common direction, such as North-East (I or II) with East, North-East (I or II) with North, South-East (I or II) and East and so on, but we can discard these possibilities too: without loss of generality we can examine the case of North-East I (II) and East. Let's suppose that A is a module that intends to occupy the grid cell with relative coordinates (1,1) by moving North-East I (II). A precondition for the movement is that the cell grid in relative position (0,1) is empty, but any module B willing to occupy the same grid cell by applying East would occupy exactly such position, and this

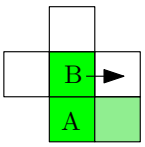


Figure 3.8: Module B wants to move East and occupy $(1, 0)$, while A cannot move to occupy the same grid cell, as it does not fulfill the preconditions of North-East I (II).

is a contradiction (see Figure 3.8). The rest of the possible mentioned conflicts present the same contradictions, and therefore can be excluded.

□

Proposition 7 *The system stays connected during the whole movement, i.e. no rule produces a disconnection between the module applying it and the system.*

Proof: Due to Lemma 3, the only cases to study are the ones originated by the application of one rule at a time. All the post-conditions of the action rules guarantee that the current module is connected to one of its neighbors (the support module) after the movement, so we can immediately discard the case in which a module disconnects from the rest of the system by the application of an action rule. The second case to study is the case in which a module that is the only connection between two connected parts of the system moves and creates a disconnection between such parts. We can reasonably assume in this proof that before the current module applies the rule the system is connected, as in the initial configuration the system is connected. To prove that such kind of disconnection cannot occur, we can examine the case of the North rule without loss of generality.

Observing the precondition of the North rule in Figure 3.3 we can notice that the module with relative coordinates $(-1, 0)$ cannot be a module, as it cannot be neither inactive nor active according to the preconditions, and it cannot be a bridge, because of the preconditions of the Bridge rule (see Figure 3.5); the only possibility for the current module to be the only connection of two different parts of the system is the situation depicted in Figure 3.9. We can easily see that this situation cannot occur. Let's suppose that module B occupying the cell grid in position $(0, -1)$ is free to move, i.e. it does not have a south neighbor; in this case, the free grid cells between module A and module B imply that even when free to apply the North-West rule, module B stopped while A continued the movement towards east, and this goes against our settings. In the case where module B has a south neighbor we can repeat the same reasoning for this neighbor. So the situation cannot occur and no disconnection is produced by the application of the North rule by one module. The rest of rules can be analyzed in an analogous way.

The last possible case of disconnection to analyze is the one in which while a module applies a rule and intends to connect to the support module, this last one applies an action rule itself, leaving its grid cell empty.

We can exclude this possibility analyzing the possible support modules, and proving that there is no rule that they could be applying.

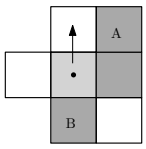


Figure 3.9: The current module disconnects the system with the application of the North rule, as it is the only connection between two different connected parts of the system.

Without loss of generality we can study the support module of the North-East I rule, and prove that it cannot apply any rule while the current module is moving. We can immediately see that the support module of the North-East I rule does not fulfill the preconditions of North, North-East I and II, East, South-East nor North-West; the only possible action rules left are South, South-West and South-East II. As the support module of North-East I needs to be inactive, we can easily exclude these possibilities too, as it can be proved that any module able to apply one of these rules has to be active, as it has already been activated by the application of previous rules. The support modules of the other rules can be studied in the same way, so this kind of disconnection cannot occur, and the system stays connected during the whole locomotion. □

Proposition 8 *There is always at least one rule that is applicable to some module of the system.*

Proof: The proof is analogous to the case of the free locomotion; we can analyze the movement of the leftmost column and proceed by induction. Among the modules of the leftmost column there is always a module able to apply a rule: either its topmost module, or the bottommost module in the case an obstacle is encountered. If the topmost is free to move, as soon as it applies the North-East I rule it leaves space to the other modules of its column to activate and move North and North-East progressively; the first module, followed by the others of its column, moves East on the top of the system, South-East, and South along the rightmost column, and deactivates. If the bottommost moves first, it applies North-West and moves North until it reaches the top of the system, followed by the other modules of the column, and continues in the same way leaving the new leftmost column free to move. As soon as these modules make their way to the right, the second leftmost column becomes the first, either its topmost or its bottommost module is free to move and the motion is repeated. As there is always a leftmost column, by induction on the number of columns we can conclude that there is always some rule applicable to some module of the system. □

Proposition 9 *The rules actually produce a locomotion of the system from west to east, on free ground and over low obstacles.*

Proof: As there is always one rule applicable to the system because of Lemma 8, the only thing we need to prove is that the rules don't produce an alternation of opposite movements without moving the system in any direction. The action rules that don't produce a movement in the East direction are North, South,

North-West and South-West; we can easily notice from the preconditions of the rules that any module which performs a movement in the North direction cannot produce one in the South direction until it has reached the opposite side of the system; the same can be said about North-west and South-West: no module can oscillate applying North-West and South-West, as any module which performs North-west has to reach the right side of the system before being able to move South-West, so we cannot have an oscillation produced by the application of these rules. As all the other action rules have a component on the East direction, the application of these rules produces an overall movement towards East, with and without low obstacles, without any risk of oscillations. □

We have proved the following:

Theorem 2 *The rules described in Section 3.3 allow any rectangular configuration of modules to advance eastwards on a free ground and in presence of low obstacles, while keeping the system connected.*

We have already commented the stop rules and their correctness in Section 2.4; in the presence of low obstacles the same rules produce a reconfiguration of the system into a rectangle at the end of the locomotion, as long as the value of the counter of the number of rounds is big enough to allow the system to traverse the obstacles before reconfiguring.

3.6 Complexity

Neighbourhood During the locomotion each module is only able to check if any grid cell sharing either an edge or a vertex with him is empty or occupied by another module; in this last case, it can obtain information about the state of such module. No information about any other position is needed.

Memory and computation Locomotion only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to memorize a fixed value in a counter, and to check if preconditions of rules are fulfilled.

Number of moves For an advance length of k rounds, and a rectangular system of dimension $h \times b$, each module performs in the worst case $O(k(2h+b))$ changes of position, $O(k)$ deactivations (it deactivates once at each round) and stops once, therefore each module performs at most $O(kn)$ rules. The number k of rounds of the system depends linearly on the number of modules of the obstacle, supposing that the system starts its locomotion just before the obstacle starts, and ends it as soon as it overpasses the obstacle.

Communication A constant size communication is performed during the application of each rule, so the communication performed by each module is linear in the number of moves, i.e. $O(kn)$.

Number of time steps Let's suppose that the execution of the rules is synchronized, with an advance of k rounds and a rectangular system of dimension $h \times b$. As in the case of free locomotion we can compute the

number of time steps by charging to each column the steps starting from the activation of the topmost module of the column to the first movement of the topmost module of the following column, and multiplying by the number of columns; if the movement of a column starts from the topmost module the number of time steps to be charged to the column is $O(2h)$, as in the case of the free locomotion; this behavior is modified when the following column activates inverting the order of activation of its module.

For any column which starts its movement from the bottommost module, the number increases to $O(3h)$, as the delay between the activation of two modules of the same column is increased in this case; these columns start to move as soon as the last module of the previous column moves North, h steps before the others. We can charge then only h steps to the previous column and $3h$ to the column which inverts the movement, obtaining an average of $O(2h)$ for column.

When the system moves on the top of the obstacle, the number of modules of each column is less than h, and the time steps to charge to each column decrease with the number of modules; locomotion over low obstacles is then performed in at most $O(k(2hb))$ steps. The number of rounds k depends linearly on the number of modules of the obstacle. Reconfiguration only requires $O(n)$ steps, so the overall number of time steps is at most $O(kn)$.

3.7 Rules

The algorithm is based on 16 different rules; before starting the locomotion, the number of rounds needs to be stored in counter C01.

Rule name: North Priority: 3 Preconditions: !SOBSTA !SSTOPP N0*1* T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV Postconditions: P0,1 SACTIV A****

- **Rule**: North
- **Priority**: 3
- **Preconditions**:
!SSTOPP: the rule is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0*1*: the rule concerns the module without a north neighbor and with a east neighbor.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A****: the module attaches afterward if was attached before, when possible.

Rule name: North-east I Priority: 3 Preconditions: !SOBSTA !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV !T1,-1,OBSTA Postconditions: P1,1 SACTIV A***1 C000 + C000 0001

- **Rule**: North-east I
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0*1*: concerns only the modules without a North neighbor and with a east neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obstacle*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: North-east II
Priority: 3
Preconditions:!SOBSTA !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV T1,-1,OBSTA !T0,-1,ACTIV !T0,-1,INACT !T0,-1,BRIDG
Postconditions:P1,1 SACTIV A***1

- **Rule**: North-East II
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0*1*: the rule concerns only the modules without a north neighbor, but with a east neighbor.
T1,0,INACT: the support module needs to exist and to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T0,-1,INACT: the module in position $(0, -1)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *bridge*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: East
Priority: 3
Preconditions: !SOBSTA !SSTOPP N0*01 !T1,-1,ACTIV !T1,-1,OBSTA !E1,-1 !T-1,0,ACTIV !T-1,0,INACT
Postconditions:P1,0 SACTIV A***1

- **Rule**: East
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0*01: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obstacle*.
!E1,-1: the grid cell in position $(1, -1)$ cannot be empty.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South Priority: 3 Preconditions: !SOBSTA !SBRIDG !SSTOPP N*1*0 T-1,-1,INACT !T1,0,INACT !T1,0,ACTIV Postconditions: P0,-1 SACTIV A**11

- **Rule**: South
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set on *stop*.
!SOBSTA: the rule is not applicable to modules set on *obstacle*.
!SBRIDG: the rule is not applicable to modules set on *bridge*.
N*1*0: concerns the modules with a west neighbor, without a module on the south.
!T1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
T-1,-1,INACT: the support module needs to exist and to be set on *inactive*.
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new west and south neighbors; if it was attached before and where still possible, it attaches to the other neighbors.

Rule name: South-east Priority: 3 Preconditions: !SOBSTA !SSTOPP N0*01 E1,-1 !T-1,0,INACT !T-1,0,ACTIV Postconditions: P1,-1 SACTIV A*1**

- **Rule**: South-east
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0*01: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1*1: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: North-west Priority: 3 Preconditions: !SOBSTA !SSTOPP N10** E-1,1 !T1,0,INACT !T1,0,ACTIV !T0,-1,INACT !T0,-1,ACTIV T0,1,INACT !T0,-1,BRIDG Postconditions: P-1,1 SACTIV A*11*

- **Rule**: North-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N10**: the rule concerns only the modules with a north and without a west neighbors.
!T1,0,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T0,-1,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T0,1,INACT: the module in position $(0, 1)$ needs to be set on *inactive*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *bridge*.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
- **Postconditions**:
P-1,1: the module moves north-west.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new east and west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South-west Priority: 3 Preconditions: !SOBSTA !SBRIDG !SSTOPP N*1*0 T-2,-1,OBSTA E-1,-1 T-1,0,INACT Postconditions: P-1,-1 SACTIV A11*1

- **Rule**: South-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
!SBRIDG: the rule is not applicable to modules set on *bridge*.
N*1*0: the rule concerns only the modules with a west neighbor and without a south neighbor.
T-2,-1,OBSTA: the module in position $(-2, -1)$ needs to be set on *obstacle*
T-1,0,INACT: the module in position $(-1, 0)$ needs to be set on *inactive*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *bridge*.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
- **Postconditions**:
P-1,-1: the module moves south-west.
SACTIV: the module changes its state to *active*.
A11*1: the module attaches to its new north, west and south neighbor; if it was attached before and still possible, it attaches to the other neighbor.

Rule name: South-east II Priority: 2 Preconditions: !SOBSTA !SSTOPP N0101 E1,-1 E0,-2 !E-1,-1 T-1,-2,OBSTA !T-1,-1,OBSTA Postconditions: P1,-1 SACTIV A*11*

- **Rule**: South-east II
- **Priority**: 2
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N0101: the rule concerns only the modules with a west and south neighbor, without a north and a east neighbor.
T-1,-2,OBSTA: the module in position $(-1, -2)$ needs to be set on *obstacle*
!T-1,-1,OBSTA: the module in position $(-1, -1)$ needs to be set on *obstacle*
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
E0,-2: the grid cell in position $(0, -2)$ needs to be empty.
!E-1,-1: the grid cell in position $(-1, -1)$ cannot be empty.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and east neighbor; if it was attached before and still possible, it attaches to the other neighbors.

Rule name: Deactivation Priority: 1 Preconditions: SACTIV Postconditions: SINACT

- **Rule**: Deactivation
- **Priority**: 1 (the priority is smaller than the priority of any action rule; modules deactivate only if no action rule is applicable).
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Bridge Priority: 4 Preconditions: SACTIV N0110 T1,0,OBSTA T1,-1,OBSTA Postconditions: SBRIDG

- **Rule**: Bridge
- **Priority**: 4
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N0110: the rule concerns only the modules with a west and east neighbor, without a north and a south neighbor.

T1,0,OBSTA: the module in position $(1, 0)$ needs to be set on *obstacle*
T1,-1,OBSTA: the module in position $(1, -1)$ needs to be set on *obstacle* -
Postconditions:
SBRIDG: the module changes its state to *bridge*.

Rule name: Stop Priority: 5 Preconditions: SINACT N001* T1,0,INACT E1,1 = C000 C001 Postconditions: SSTOPP

- **Rule**: Stop
- **Priority**: 5 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N001*: concerns the modules with a east neighbor and without a module on the north and on the west. The presence of a south neighbor is not important.
T1,0,INACT: the east neighbor needs to be an inactive module.
E1,1: the grid cell of relative coordinates 1,1 needs to be empty.
= C000 C001: the value of the counter C000 needs to meet the value of the counter C001, this last one fixed before starting the locomotion. This last precondition controls the number of rounds.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop 1 Priority: 5 Preconditions: SINACT N*1** T-1,0,STOPP Postconditions: SSTOPP

- **Rule**: Stop 1
- **Priority**: 5 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N*1**: the module needs to have a east neighbor T-1,0,STOPP: the east neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 2 Priority: 5 Preconditions: SINACT N1*** T0,1,STOPP Postconditions: SSTOPP

- **Rule**: Stop 2
- **Priority**: 5 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N1***: the module needs to have a north neighbor T0,1,STOPP: the north neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 3 Priority: 5 Preconditions: SINACT N11** T-1,1,STOPP Postconditions: SSTOPP

- **Rule**: Stop 3
- **Priority**: 5 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
N11**: the module needs to have a north neighbor and a west neighbor.
T-1,1,STOPP: the grid cell with relative coordinates $(-1, 1)$ needs to be occupied by a module set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop 4 Priority: 5 Preconditions: SINACT N*10* T-1,0,STOPP Postconditions: SSTOPP

- **Rule**: Stop 4
- **Priority**: 5 (the priority is greater than the priority of any action rule).
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
N*10*: the module needs to have a west neighbor and no module on the east.
T-1,0,STOPP: the west neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: North-west version II Priority: 3 Preconditions: !SOBSTA !SSTOPP N101* E-1,1 T1,0,OBSTA !T0,-1,INACT !T0,-1,ACTIV T0,1,INACT Postconditions: P-1,1 SACTIV A*11*

- **Rule**: North-west version II
- **Priority**: 3
- **Preconditions**:
!SSTOPP: it is not applicable to modules set in the *stop* state.
!SOBSTA: the rule is not applicable to modules set in the *obstacle* state.
N101*: the rule concerns only the modules with a north and a east neighbor, and without a west neighbor.
T1,0,OBSTA: the module in position $(1, 0)$ need to be set on *obstacle*.
!T0,-1,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T0,1,INACT: the module in position $(0, 1)$ needs to be set on *inactive*.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
- **Postconditions**:
P-1,1: the module moves north-west.

SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new east and west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Chapter 4

Rectangle locomotion over high obstacles

4.1 Goal

In this chapter we discuss the eastward locomotion of a rectangular system in the presence of high obstacles. In this case obstacles are configured as histograms, and are not subject to height restrictions: we call them "high obstacles" in order to differentiate them from the low obstacles of the previous chapter.

The rules are divided into locomotion rules and reconfiguration rules, and we will not discuss here the reconfiguration of the system again, as it is performed as in the other cases.

4.2 Strategy

Before reaching an obstacle, the rectangle moves under the original locomotion rules discussed in Chapter 2: modules always move from the back of the group, over the top, and down on the front of the group. As the system encounters an obstacle, the behavior of the modules depends on the vertical height of the part of the obstacle that they are traversing; in the parts that do not reach the initial height of the rectangle, the strategy is exactly the same as in the case of low obstacles discussed in Chapter 3: the system fills the free spaces and flows over until it reaches the ground another time, maintaining its initial height in every moment. As soon as the height of the obstacle reaches or exceeds the initial height of the system, such strategy is not applicable anymore without generating disconnections; in this cases instead, the system continues its locomotion creating what we call a “worm” of inactive modules over the obstacle, and making active modules walk over it; we can observe an example of such strategy in Figure 4.1. As in the case of low obstacles, the formation of bridges during the locomotion allows the system to avoid filling the bottlenecks of width 1 formed between the obstacle and the system itself (refer to Chapter 3). Such bridges are formed in the same way both in the low and the high parts of the obstacle. Once the obstacle is overpassed, the system recovers its initial shape and stops the locomotion through the usual stop rules.

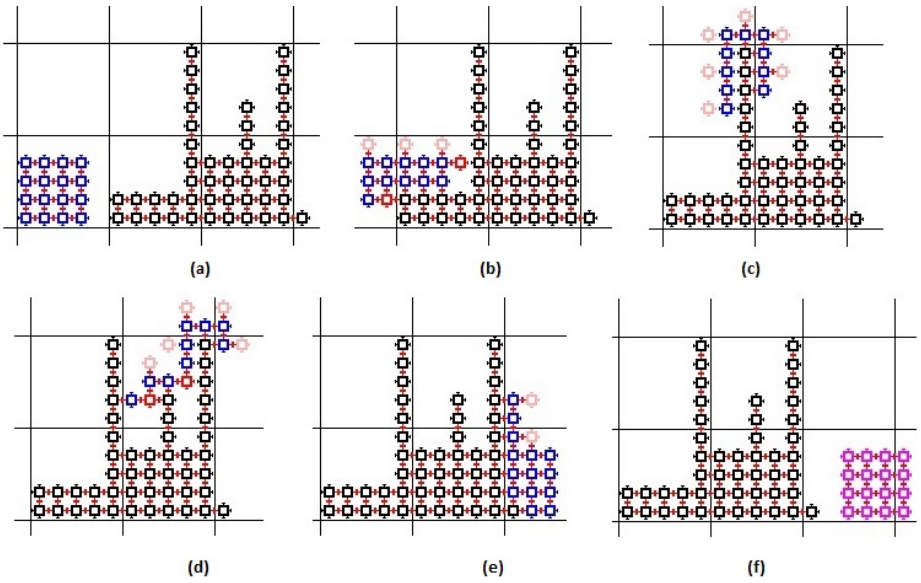


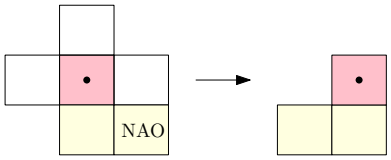
Figure 4.1: Crawling over a high obstacle: the system starts configured as a rectangle; in (b) we can see that as the first part of the obstacle does not reach the height of the system, and the overpassing is done as in the low obstacles case. In (c) and (d) the system changes its shape in order to overpass the high parts of the obstacle; the last part is low, so the system in (e) recovers its shape, and stops the locomotion reconfiguring in (f). Active modules are depicted in pink, inactive modules are blue, obstacles are black and bridges are the red modules.

While the modules are configured as a worm over an high part of an obstacle, the fact that in the initial configuration the system was formed by one or by more than one column is not relevant; the modifications to be done for the case of one column are then analogous to the modifications applied for the overpassing of low obstacles, and we do not discuss them here again.

4.3 Locomotion rules

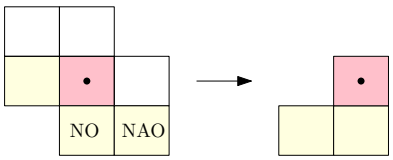
The locomotion of the system in the presence of high obstacles is mostly performed through the application of the rules used in the low case, with some adjustments and additions that deal with the high parts. The distinction between high and low obstacles is obtained through the use of counters; before the locomotion starts, a value is stored in the counter C00 of each module, representing its vertical position, measured considering the line of the ground as the x-axis; each time a movement with a north (south) component is performed, such counter is increased (decreased) by 1. Moreover, the height of the system is stored into the C02 counter of each module before the movement starts. Such counters are used in order to distinguish, in some cases, which of the rules are applicable in the low parts of the obstacles and which are applicable only in the

high parts, while the system is configured as a worm. Examples of such method are the two east rules depicted in Figure 4.2.



East I

East I
3
State≠(Obsta,Bridge,Stop)&(0,1)=(-1,0)=(1,0)=empty&(0,-1)=(1,-1)=full&State(1,-1)≠(Active,Obsta)
Move to (1,0) & State = (Active)



East II

East II
3
State≠(Obsta,Bridge,Stop)&(0,1)=(1,0)=empty&(0,-1)=(-1,0)=(1,-1)=full&State(1,-1)≠
(Active,Obsta)&State(-1,1)≠Active&State(0,-1)≠Obsta&Not(State(-1,-1)=Obsta&State(-1,0)≠Obsta)
& C00 > C02
Move to (1,0) & State = (Active)

Figure 4.2: East rules for the locomotion with high obstacles: the East I rule is analogous to the East rule of low obstacles, and is applicable for any height; the East II rule deals with the new situations that can be generated in the high parts of the obstacle, and it can be applied only in such cases (see the precondition C00 > C02). Active modules are pink, empty squares are empty grid cells, and colored grid cells are full positions; NAO indicates that if a module occupies the grid cell it cannot be set in the active nor in the obstacle state. Notice that the most complex preconditions are only presented in the details of the text rules and not in their graphic representation.

The reconfiguration of the rectangle in the high part of the obstacle is avoided through the application of the same method: we can notice in Figure 4.3 that the two depicted south-east rules are only applicable to the case of $C00 > C02$; a module with a lower height cannot apply the rules, and deactivates over its south neighbor, while a module which is walking on a high part of the obstacle continues its locomotion moving south-east.

The formation of the worm over the high parts is achieved through the deactivation rules, depicted in Figure 4.4. The details of the rest of the rules are presented in Section 4.6.

4.4 Correctness

The modifications applied to the previous rules for the low obstacles are operated in order to adapt the same rules to the new states of the modules and to some particular new situation; all the results proved in the previous chapter are still valid for the new version of the rules, and we will not prove them again here. In this section we will use such results and examine the impact of the new defined rules on the correctness conclusions.

As always, we will prove that:

1. The system stays connected during the whole movement.
2. The new rules do not produce collisions during the movement.
3. There is always at least one rule that is applicable to some module of the system.
4. The rules produce a locomotion of the system from West to East in the presence of high obstacles.

Proposition 10 *The system stays connected during the whole movement, i.e. no rule produces a disconnection between the module applying it and the system.*

Proof: We have already proved that the free locomotion and the low obstacle rules do not generate any disconnection; we want now to exclude the case of a disconnection generated during the overpassing of the high parts obstacle.

We analyze the *South-East II* rule depicted in Figure 4.3, and assume that the system is connected before the application of the rule. As any module able to apply this rule has a South neighbor, and does not have any North nor East neighbor, the only possibility for the module to create a disconnection is that the grid cell in relative position $(-1, 0)$ is occupied by a module of the system, and that the current module is the only connection between such East neighbor and its South neighbor. This is possible only in the case in which the relative position $(-1, -1)$ is either empty or occupied by an obstacle. The case of the position $(-1, -1)$ being empty can be easily discarded, as such configuration cannot be reached by a rectangular system which locomotes under our strategy of movement, as in our setting modules continue their locomotion until they find either an obstacle or the ground blocking their way. The second possibility is explicitly excluded by the precondition *!(!T-1,0,OBSTA !E-1,0 T-1,-1,OBSTA)*. Such contradictions can be found in all the new rules, so no disconnections are produced. □

Lemma 4 *Each module can apply at most one action rule at a time.*

Proof: We have already proved in the previous chapter that the set of rules that produce the overpassing of low obstacles satisfies this property; the only thing to check now is that none of the new action rules interferes with the preexistent rules, nor with another new rule. In order to prove this result, we can analyze the details of the new advance rule North II depicted in Figure 4.5 and prove that it cannot interfere with any other rule. The rest of rules can be treated in the same way. We can easily see that any module that satisfies the preconditions of North II cannot perform any movement in the North-East,

East, South-East nor South direction, due to the presence of modules blocking the way; the North-West rule requires the presence of a North neighbor, and the South-East rule is applicable only if the grid cell in position $(0, -1)$ is empty, so they are both incompatible with North II. As there is no rule that produces a movement towards west, the only rules left to check are North I and North II; North I is applicable only by modules without a South neighbor, and North III requires the grid cell in relative position $(1, -1)$ to be empty; we can conclude then that any module able to apply such rule cannot satisfy the preconditions of any other. As all the rules present such properties, we can conclude that each module can apply at most one action rule at a time. □

Proposition 11 *No collisions are produced during the movement.*

Proof: We want to prove that two different modules never move to the same grid cell at the same time while applying the rules; the proof is analogous to the case of low obstacles. Due to Lemma 4 we only need to worry about conflicts created by the application of one action rule at a time. The case of conflicts between the North and the South rules can be easily excluded, as in the case of the free locomotion and of low obstacles: any module able to apply North is a module attached to the left side of the system, while any module applying South is attached to the right, and the two movements cannot conflict, as we avoid entering in bottlenecks of width 1. The case of conflicts generated by rules which produce the same change of position as North-East I and North-East II, or as South-East I and South-East II, is clearly impossible: two modules willing to move to the same grid cell performing the same movement need to occupy the same position in the first place, and this is not allowed in our configuration. The only other type of conflict which could occur is the one generated by rules with a common direction, such as North-East (I or II) with East, North-East (I or II) with North (I,II,III), South-East (I or II) and East (I or II) and so on, but we can discard these possibilities too: without loss of generality we can examine the case of North-East I (II) and East I (II). Any module that intends to occupy the grid cell with relative coordinates (1,1) by moving North-East I (II) cannot have a North neighbor; any module willing to occupy the same grid cell by applying East would occupy exactly the North neighbor position, and this is a contradiction. The rest of the possible mentioned conflicts present the same contradictions, and therefore can be excluded. □

Proposition 12 *There is always at least one rule that is applicable to some module of the system.*

Proof: We have already proved in the previous chapters that during the free locomotion and the overpassing of low obstacles there is always a module that is able to apply some rule, so the modules are always able to reach the high parts of the obstacle at some moment. On the high part, in the same way, there is always a module able to apply a rule: this is either an active module walking on the inactive worm, either an active module which inactivates in front of the worm, either is the tail module of the worm, able to activate again as all the active modules of the system have passed over it. □

Proposition 13 *The rules actually produce a locomotion of the system from west to east, on free ground and over histogram obstacles.*

Proof: As there is always one rule applicable to the system because of Proposition 12, the only thing we need to prove is that the rules do not produce an alternation of opposite movements without moving the system in any direction. We have already proved in chapters 2 and 3 that the free locomotion and the locomotion over low obstacles is correctly performed in this sense, and we prove now that there is no oscillation produced by the new rules introduced. As an example of pair of rules which could produce an oscillation of the system, we analyze the pair North (I, II or III) and South. We can easily see that any module which performs a movement in the North direction through the application of a North rule cannot produce one in the South direction until it has reached the opposite side of the column of the obstacle which is being overpassed, as it is clearly shown by the preconditions of the rules (see Section RulesRecthigh for the detailed rules). All the other pairs present the same behavior and can be analyzed in the same way. □

4.5 Complexity

Neighbourhood During the locomotion each module is only able to check if any grid cell sharing either an edge or a vertex with it is empty or occupied by another module; in this last case, it can obtain information about the state of such module. No information about any other position is needed. In some situations the same information is needed for some of the grid cells of the second neighbor; such extension is restricted to the case in which the stability of the system could be compromised.

Memory and computation Locomotion only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to memorize a fixed value in a counter, to perform some simple operations with counters, and to check if preconditions of rules are fulfilled.

Number of moves As we have already analyzed the case of the low obstacles, we can now focus on the overpassing of the high parts.

As modules deactivate one by one in a worm shape during the overpassing of the high parts, the worst case possible for the number of moves of a module, from the moment of activation to the moment of inactivation, is the case in which the rest of $n-1$ modules are all inactive forming the worm, and the current module needs to walk over each one of them before being able to inactivate again; so, in the worst case, a module performs $O(n)$ moves during this phase of the locomotion. The overall number of moves performed by a module during the overpassing of the obstacle is $O(kn)$, where k is the number of rounds performed by the system during the locomotion, and is directly proportional to the number M of the modules of the obstacle.

Communication A constant size communication is performed during the application of each rule, so the communication performed by each module is $O(kn)$.

Number of time steps Given the results of the low obstacle case, we can now ideally divide the obstacle into its high and low parts, and study separately

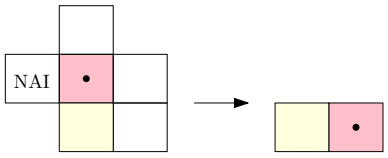
the number of moves needed for the overpassing of the high ones; we suppose here, as always, that the execution of the rules is synchronized.

In Figure 4.6 we can see an example of an high obstacle ideally divided into its low and high part; we recall that an obstacle is considered low if its height is at most $h-1$, where h is the initial height of the rectangle. In the computation we charge to the high part of the obstacles all the moves made in order to reach and overpass grid cells with vertical height bigger than h. In particular then, we need to consider all the moves applied to reach, occupy and clear all the grid cells forming the boundary of the high modules of the obstacle that need to be overpassed due to the presence of the high part (the grid cells numbered from 1 to M in Figure 4.6). Let M be the number of grid cells of the boundary of the high part in exam; if we charge to each module of the system the time steps starting from its deactivation in the worm to the inactivation of the following module of the worm, we can easily see that, as module moves together with a distance of 1 grid cell, the number of time steps needed in order to create a worm over such part of the obstacle is $O(M)$, independently from the number of modules of the system.

If the number n of modules of the system is less than M, an additional $O(n)$ time steps will be needed in order to clear such cells and complete the traversing.

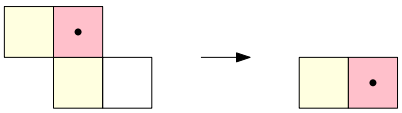
If the number of modules is bigger than M, an additional $O(n - M)$ number of time steps is needed as, before starting the reactivation of its modules, the worm needs to wait until all the modules that are not part of the worm have walked over.

As in the case of low obstacles then, the overall number of time steps is directly proportional to the number n of modules of the system and to the number of modules of the obstacle; this last dependency is reflected in the number k of complete rounds that the system needs to perform in order to overpass the obstacle.



South-east I

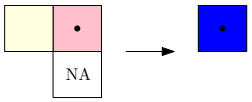
South-east I 6 State≠(Obsta,Bridge,Stop)&(0,1)=(1,0)=(1,-1)=empty&(0,-1)=full&State(-1,0)≠(Active,Inact)&C00>C02 Move to (1,-1) & State=Active&Add -1 to C00



South-east II

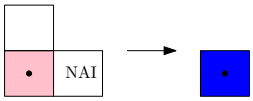
South-east II 5 State≠(Obsta,Bridge,Stop)&(0,1)=(1,0)=(1,-1)=empty&(0,-1)=full&State(0,-1)≠Obsta &Not(State(-1,0)≠Obsta&(-1,0)=empty&State(-1,-1)=Obsta) & C00 > C02 Move to (1,-1) & State = Active& Add -1 to C00

Figure 4.3: South-East rules for the locomotion with high obstacles: the South-East I rule is analogous to the South-East rule of low obstacles, with the additional condition $C00 > C02$; the South-East II rule deals with the new situations that can be generated in the high parts of the obstacle; both rules, as we can notice, can be applied only in the case C00 $>$ C02. Active modules are pink, empty squares are empty grid cells, and colored grid cells are full positions; NAI indicates that if a module occupies the grid cell it cannot be set in the active nor in the inactive state. In this case too some preconditions are only presented in the details of the text version of the rules as they are difficult to depict in their graphic representation.



Deactivation I

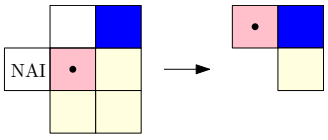
Deactivation I
2
State=Active&(-1,0)=full&State(0,-1)≠Active
State=Inact



Deactivation II

Deactivation II
1
State=Active&(0,1)=empty&State(1,0)≠(Active,Inact)
State=Inact

Figure 4.4: Deactivation rules for the locomotion with high obstacles: the rules produce both the deactivation in the reconfiguration of the rectangle, in the free locomotion and in the low obstacle, and the formation of the worm on the higher parts. As always, active modules are pink, blue modules are inactive, empty squares are empty grid cells, and colored grid cells are full positions; NAI indicates that if a module occupies the grid cell it cannot be set in the active nor in the inactive state.



North II

North II
3
State≠(Obsta,Bridge,Stop)&(0,1)=empty&(1,-1)=(1,0)=full&State(1,1)=Inact&State(-1,0)≠(Inact,Active)
Move to (0,1)&State=Active&Add 1 to counter C00

Figure 4.5: Details of the North II rule.

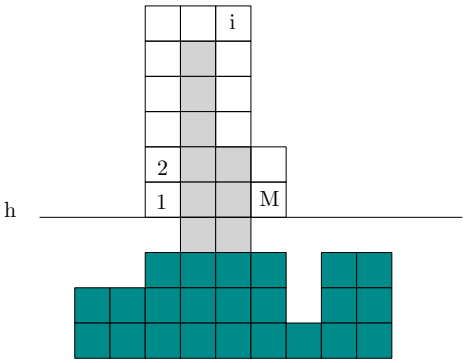


Figure 4.6: Time steps for the overpassing of an high obstacle: the obstacle can be ideally divided into the low part (cyan in the Picture) and the high part (gray in the Picture); the horizontal line represents the initial height h of the system. The empty grid cells numbered from 1 to M are the boundary cells of the high part of the obstacle.

4.6 Rules

The algorithm is based on 20 different rules; before starting the locomotion each module has the following information stored in its counters: the vertical heigh of the module (C01), the initial heigh of the system (C02) and the number of rounds (C03).

Rule name: North I
Priority:3
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0*10 T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV
Postconditions:P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North I
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0*10: the rule concerns the module without a north neighbor and with a east neighbor.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North II
Priority:3
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0*11 !E1,-1 !(T1,-1,OBSTA !T0,-1,OBSTA) T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV
Postconditions:P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North II
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0*11: the rule concerns the module without a north neighbor and with a east and a south neighbor.
!E1,-1: the grid cell in position $(1, -1)$ needs to be occupied.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!(T1,-1,OBSTA !T0,-1,OBSTA): the rule is not applicable when the module in position $(1, -1)$ is an obstacle and the module in $(0, -1)$ is not an obstacle.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North III
Priority:3
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0*11 E1,-1 T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV T0,-1,OBSTA
Postconditions:P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North III
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0*11: the rule concerns the module without a north neighbor and with a east and a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
T0,-1,OBSTA: the module in position $(0, -1)$ needs to be set on *obstacle*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North-east I
Priority:3
Preconditions:!SOBSTA !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV !T1,-1,OBSTA
Postconditions:P1,1 SACTIV A**11 C000 + C000 0001 C004 + C004 0001

- **Rule**: North-east I
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA: it is not applicable to modules set in the *stop* or *obstacle* state.
N0*1*: concerns only the modules without a North neighbor and with a east neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obstacle*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.
C004 + C004 0001: the counter C004 is increased by 1 each time the rule is applied.

Rule name: North-east II
Priority:3
Preconditions:!SOBSTA !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV T1,-1,OBSTA !T0,-1,ACTIV !T0,-1,INACT
Postconditions:P1,1 SACTIV A**11 C000 + C000 0001

- **Rule**: North-east II
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA: it is not applicable to modules set in the *stop* or *obstacle* state.
N0*1*: concerns only the modules without a North neighbor and with a east neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T0,-1,INACT: the module in position $(0, -1)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
T1,-1,OBSTA: the module in position $(1, -1)$ needs to be set on *obstacle*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: East I
Priority:3
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0101 !T1,-1,ACTIV !T1,-1,OBSTA !T-1,1,ACTIV !E1,-1 !T0,-1,OBSTA !(T-1,-1,OBSTA !T-1,0,OBSTA) ¿ C000 C002
Postconditions:P1,0 SACTIV A**11

- **Rule**: East I
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0101: the rule concerns only the modules without north or west neighbors, but with a west and a south neighbor.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obstacle*.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!E1,-1: the grid cell in position $(1, -1)$ cannot be empty.
!T0,-1,OBSTA: the module in position $(0, -1)$ cannot be an obstacle.
!(T-1,-1,OBSTA !T-1,0,OBSTA): the rule is not applicable when the module in position $(-1, -1)$ is an obstacle and the module in $(-1, 0)$ is not an obstacle.
$>$ C000 C002: the value of the counter C00 needs to be bigger than the value of C02.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.

A**11: the module attaches to its new south and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: East II Priority:3 Preconditions:!SOBSTA !SBRIDG !SSTOPP N0001 !T1,-1,ACTIV !T1,-1,OBSTA !E1,-1 Postconditions:P1,0 SACTIV A***1

- **Rule**: East II
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0001: the rule concerns only the modules without north, west or east neighbors, but with a south neighbor.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obstacle*.
!E1,-1: the grid cell in position $(1, -1)$ cannot be empty.
!T1,-1,OBSTA: the module in position $(1, -1)$ cannot to be an obstacle.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot to be active.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South Priority:3 Preconditions:!SOBSTA !SBRIDG !SSTOPP N*1*0 T-1,-1,INACT !T1,0,INACT !T1,0,ACTIV Postconditions:P0,-1 SACTIV A**11 C000 - C000 0001

- **Rule**: South
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N*1*0: concerns the modules with a west neighbor, without a module on the south.
!T1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
T-1,-1,INACT: the support module needs to exist and to be set on *inactive*.
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new west and south neighbors; if it was attached before and where still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name:South-east I
Priority:6
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0*01 E1,-1 !T-1,0,ACTIV !T-1,0,INACT > C000 C002
Postconditions:P1,-1 SACTIV A*1** C000 - C000 0001

- **Rule**: South-east I
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0*01: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
> C000 C002: the value of the counter C00 needs to be bigger than the value of C02.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1*1: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east II
Priority:5
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0*01 E1,-1 !T0,-1,OBSTA
!(!T-1,0,OBSTA !E-1,0 T-1,-1,OBSTA) > C000 C002
Postconditions:P1,-1 SACTIV A*11* C000 - C000 0001

- **Rule**: South-east II
- **Priority**: 5
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0*01: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
!T0,-1,OBSTA: the module in position $(0, -1)$ cannot be an obstacle.
!(!T-1,0,OBSTA !E-1,0 T-1,-1,OBSTA): the rule is not applicable when the grid cell in position $(-1, 0)$ is not occupied by an obstacle nor empty, and the module in $(-1, -1)$ is an obstacle.
> C000 C002: the value of the counter C00 needs to be bigger than the value of C02.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1*1: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east III
Priority:3
Preconditions:!SOBSTA !SBRIDG !SSTOPP N0101 E1,-1 E0,-2 T-1,-2,OBSTA
!(!T-1,0,OBSTA T-1,-1,OBSTA)
Postconditions:P1,-1 SACTIV A*11* C000 - C000 0001

- **Rule**: South-east III
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N0101: the rule concerns only the modules without north or west neighbors, but with a south and a east neighbor.
E0,-2: the grid cell in position $(0, -2)$ needs to be empty.
!T-1,-2,OBSTA: the module in position $(-1, -2)$ cannot be an obstacle.
!(!T-1,0,OBSTA T-1,-1,OBSTA): the rule is not applicable when the grid cell in position $(-1, 0)$ is not occupied by an obstacle nor empty, and the module in $(-1, -1)$ is an obstacle.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: North-west
Priority:3
Preconditions:!SOBSTA !SSTOPP N10** E-1,1 !T1,0,INACT !T1,0,ACTIV !T0,-1,INACT !T0,-1,ACTIV T0,1,INACT !T0,-1,BRIDG
Postconditions:P-1,1 SACTIV A*11* C000 + C000 0001

- **Rule**: North-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N10**: the rule concerns only the modules with a north neighbor, but without a west neighbor.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T1,0,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T0,-1,INACT: the module in position $(0, -1)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *active*.
T0,1,INACT: the module in position $(0, 1)$ needs to be set on *inactive*.
- **Postconditions**:
P-1,1: the module moves north-west.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: South-west
Priority:3
Preconditions:!SOBSTA !SSTOPP !SBRIDG N*1*0 T-2,-1,OBSTA E-1,-1 T-1,0,INACT
Postconditions:P-1,-1 SACTIV A1111 C000 - C000 0001

- **Rule**: South-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTA !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge* or *obstacle* state.
N*1*0: the rule concerns only the modules without a south neighbor, but with a west neighbor.
E-1,-1: the grid cell in position $(-1,-1)$ needs to be empty.
T-1,0,INACT: the module in position $(-1,0)$ needs to be set on *inactive*.
T-2,-1,OBSTA: the module in position $(-2,-1)$ needs to be set on *obstacle*.
- **Postconditions**:
P-1,-1: the module moves south-west.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new south, east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: Deactivation I
Priority:2
Preconditions:SACTIV N*1** !T0,-1,ACTIV
Postconditions:SINACT

- **Rule**: Deactivation I
- **Priority**: 2
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N*1**: the rule concerns only the modules with a west neighbor.
!T0,-1,ACTIV: the module in position $(0,-1)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Deactivation II
Priority:1
Preconditions:SACTIV N0*** !T1,0,INACT !T1,0,ACTIV
Postconditions:SINACT

- **Rule**: Deactivation II
- **Priority**: 1
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N0***: the rule concerns only the modules without a north neighbor.
!T1,0,INACT: the module in position $(1,0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(1,0)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Formation of bridges
Priority:6
Preconditions:SACTIV N0110 T1,0,OBSTA
Postconditions:SBRIDG

- **Rule**: Formation of bridges
- **Priority**: 6
- **Preconditions**:
SACTIV: the module changes its state to *active*.
N0110: the rule concerns only the modules without a north and south neighbor, but with a west and a east one.
T1,0,OBSTA: the module in position $(1, 0)$ needs to be set on *obstacle*.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.

Rule name: Stop I
Priority:10
Preconditions:SINACT N0011 = C004 C003 = C000 C002
Postconditions:SSTOPP

- **Rule**: Stop I
- **Priority**: 10
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N0011: the rule concerns only the modules without a north and east neighbor, but with a west and a south one.
= C004 C003: counter C04 needs to meet the value of C03.
= C000 C002: counter C00 needs to meet the value of C02.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop II
Priority:10
Preconditions:SINACT T-1,0,STOPP
Postconditions:SSTOPP

- **Rule**: Stop II
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T-1,0,STOPP: the module in position $(-1, 0)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop III
Priority:10
Preconditions:SINACT T0,1,STOPP
Postconditions:SSTOPP

- **Rule**: Stop III
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T0,1,STOPP: the module in position $(0, 1)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop IV Priority:10 Preconditions:SINACT T-1,-1,STOPP Postconditions:SSTOPP

- **Rule**: Stop IV
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T-1,-1,STOPP: the module in position $(-1,-1)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Chapter 5

Rectangle locomotion under superior obstacles

5.1 Goal

In this chapter we discuss the eastward locomotion of a modular robotic system initially configured as a rectangle in the presence of superior obstacles.

We call *superior* an obstacle formed by connected modules hanging down from the "ceiling"; the minimum distance between the obstacle and the floor is of two grid cells.

As in the free locomotion case, the rules are divided into locomotion and reconfiguration rules.

5.2 Strategy

The strategy is analogue to the one of the free locomotion, but with some modifications that make it possible to crawl under the obstacles.

Before they encounter the obstacle, modules move from the back of the group, over the top, and down on the front of the group, reconfiguring into the original shape at each round. As the presence of an obstacle is detected, the system gradually flattens as much as it is necessary in order to be able to crawl and reach its right part; under the obstacle, columns are reconfigured entirely when possible, and divided into different subcolumns if the total configuration is not achievable due to the height restrictions.

We can observe an example of the overpassing of a superior obstacle in Figure 5.1. As the overpassing is complete, the system reconfigures into the initial shape and stops the locomotion.

We can notice how the same rules produce the overpassing of superior general obstacles; the system does not detect any difference between the case of an histogram and a general shaped obstacle, as the rules do not use, nor explicitly nor in an implicit way, the fact that the obstacle modules are configured as an histogram. In Figure 5.2 we can see an example of the overpassing of a general obstacle.

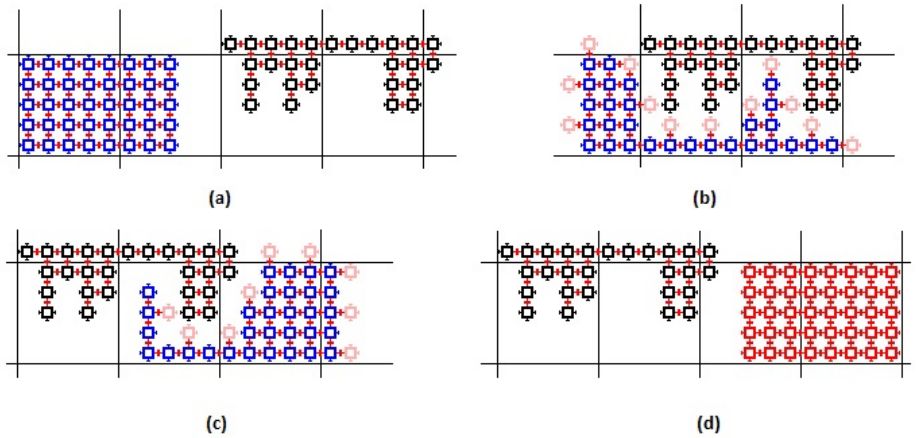


Figure 5.1: Crawling under a superior obstacle configured as an histogram: in (a) we can see that the system is configured as a rectangle when it starts the locomotion. As the obstacle is detected, in (b) and (c) modules move south-east, flattening the system and decreasing its height in order to be able to pass by. After the overpassing of the obstacle, the system reconfigures again and stops the locomotion in (d). Active modules are depicted in pink, inactive modules are blue, obstacles are black and stopped modules are red.

5.3 Advance rules

In order to perform locomotion in the presence of superior obstacles, modules apply a set of six action rules. The locomotion of the system before the overpassing of the obstacle is mostly produced through the five advance rules used in the free locomotion of the rectangle; such rules are adjusted to the introduction of the new *obstacle* state, and to the new configurations that can be produced by the presence of a superior obstacle; some operations with counters are introduced too in order to control the position of each module; an example of such modifications can be observed in the details of the North rule, depicted in Figure 5.3.

As the obstacle is encountered, an additional rule is needed in order to flatten the system and decrease its height: the South-East I rule; such rule prevents the modules from inactivating where they could create bottlenecks with the obstacle, and forces them to continue the movement, avoiding the complete reconfiguration of the columns when it is not possible; we can see its details in Figure 5.4.

As in the locomotion case, the advance rules created for the general rectangle need to be modified in the case of a rectangular system formed by only one column of modules.

In this case, to produce the locomotion of the system is enough to modify the South-East rule to avoid disconnections.

The second version of the South-East rule is depicted in Figure 5.5.

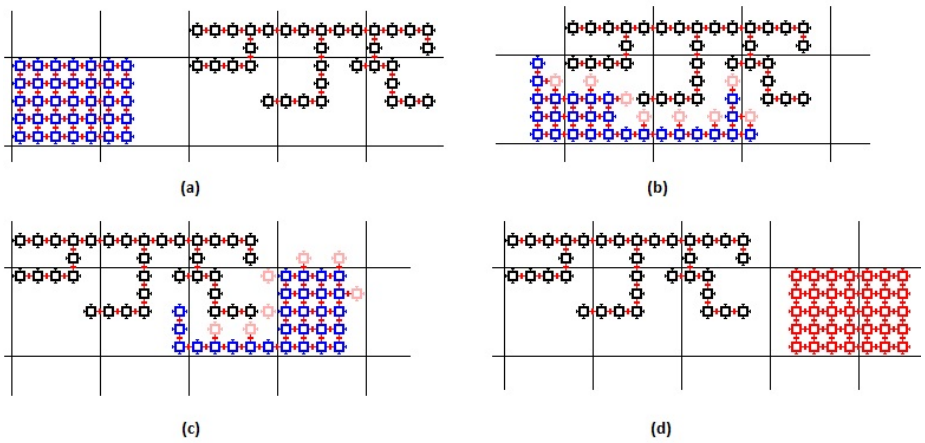
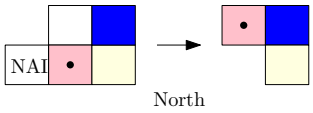


Figure 5.2: Crawling under a superior general obstacle: the system behaves as in the previous cases. Inactive modules are blue, active modules are pink, stopped modules are red and obstacles are black.



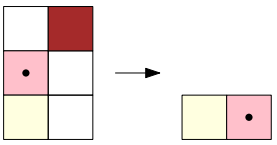
North
4
State≠(Obstacle,Stop)&(0,1)=empty&(1,0)=Full&State(-1,0)≠(Inactive,Active)&State(1,1)=Inactive
Move to (0,1)&State=Active&Add 1 to counter C00

Figure 5.3: Detailed example of the structure of the North rule.

5.4 Overpassing the obstacle

Before the locomotion starts, the value of the counter C00 is stored in the memory of each one of the modules of the system, indicating the vertical position of the module (if we consider the line representing the ground as the x-axis). Each time a module performs a rule that produces a movement with a component in the north or the south direction, the C00 counter is modified, adding 1 when its vertical height is increased and subtracting 1 when it decreases, as we can observe in the postconditions of the North rule in Figure 5.3. Moreover, the counter C01 is fixed before the locomotion, and stores the initial height of the rectangle. The complete or partial reconfiguration of the columns during the free locomotion and during the overpassing of the obstacle makes use of these counters, and is obtained through the application of two deactivation rules. The first deactivation rule is a simple adaptation of the deactivation rule used in the free locomotion of the obstacle; we can see it depicted in Figure 5.6.

The Deactivation II rule is introduced in order to produce the correct recon-



South-east II

South-east II 5 State≠(Obstacle,Stop)&(0,1)=empty&(1,0)=empty&(0,-1)=full&(1,-1)=empty&State(1,1)=Obstacle Move to (1,-1)&State=Active&Add 1 to counter C00

Figure 5.4: As the obstacle is detected, the module applies the South-West II rule and continues the movement, without blocking the way to the others. The current module is depicted in pink, the obstacle in red; full grid cells are colored and empty grid cells are not.

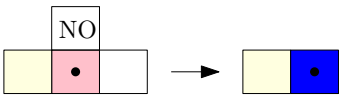
South-west II 4 State≠(Obstacle,Stop)&(-1,0)=empty&(1,0)=empty&(0,-1)=full&(1,-1)=empty&State(0,1)≠ Active& State(0,1)≠Inactive&State(-1,-1)≠Inactive&Not(State= Inactive&(-1,-1)=full) Move to (1,-1)&State=Active&Add -1 to counter C00

Figure 5.5: Advance rules: second version of the South-East rule for the case of one column.

figuration of the columns during the overpassing of the obstacle; its details are presented in Figure 5.7; this rule, when possible, produces the reconfiguration of a column with the same height as the initial height of the rectangle; when a deactivation implies a block of the way for the rest of the modules the rule is not applicable, so the columns only reconfigure partially in order to let the locomotion continue.

5.5 Reconfiguration

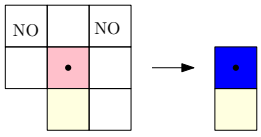
We recall that the reconfiguration strategy of the free locomotion case is based on the counting of the rounds of the system; such rounds are controlled by the counter C04, which counts the number of times each module applies the North-East rule. In the case of the free locomotion, this counter is indicative for the rounds of the system, as modules of the same column cross the system by the application of the North-East rule the same number of times during the locomotion, and no differences can be produced between modules of the same column. In the case of superior obstacles, however, it can happen that modules of the same column apply such rule a different number of times, due to the divisions into subcolumns operated by the system while overpassing the obstacle. As the system always conserves its rectangular shape during the locomotion, the idea of the reconfiguration is to stop the locomotion after a given horizontal position is reached by the leftmost column of the system.



Deactivation

Deactivation
6
State=Active&(1,0)=empty&(0,1)=full&State(0,1)≠Obstacle
State = Inactive

Figure 5.6: First deactivation rule: the condition *NO* (Not an Obstacle) is introduced for the grid cell in relative position $(0, 1)$ in order to deal with the new possible configurations; notice that we do not have the same precondition for the relative position $(1, 1)$; this is because the South-East II rule has a higher priority than the Deactivation rule, so a module which encounters a bottleneck will always move, even if the Deactivation preconditions are fulfilled.



Deactivation II

Deactivation II
6
State=Active&(0,1)=(-1,0)=(1,0)=(1,-1)=empty&(0,-1)=full&State(-1,1)≠Obstacle&State(1,1)≠Obstacle&C00≤C01
State = Inactive

Figure 5.7: Second deactivation rule: the condition *NO* (Not an Obstacle) for grid cells $(-1, 1)$ and $(1, 1)$ and the lack of a north neighbor prevent the current module to inactivate and block the bottlenecks formed between the system and the obstacle; the condition C00≤C01 permits the reconfiguration of the columns only until the initial height of the rectangle is reached.

The counters used for the new reconfiguration process are the counters C03 and C04. The counter C03 is a measure of the position of the modules on the x-axis; initially set as equal to the column counter (1 for the first column, 2 for the second and so on), C03 is increased by one each time a movement with an east component is performed; the position of each module is then controlled by its horizontal distance from the initial position of the first column of the system.

The C04 counter, fixed before the locomotion starts, stores the x-axis position in which the system will start its reconfiguration. As soon as the counter C03 of the topmost module of the leftmost column is bigger than C04, such module sets its state to *stop*, blocking the locomotion.

5.6 Correctness

As the strategy of the movement produced by the set of rules presented in this chapter is analogous to the one of the free locomotion, all the results proved in Chapter 2 can be repeated in an analogous way for this new case.

Notice that the only difference between the free locomotion and the overpassing of a superior obstacle is that when the system detects the presence of the obstacle it reconfigures its modules into smaller columns; it is easy to see that this modification cannot generate any conflict nor disconnection, as long as the priorities and preconditions are correctly adapted to the new configurations.

As in the case of the free locomotion, we can then state that:

Theorem 3 *The rules described in Section 5.3 allow any rectangular configuration to move eastward on free ground and in the presence of superior obstacles, without producing any disconnection of the system.*

5.7 Complexity

Neighbourhood As in the case of the free locomotion, each module is only able to check if any grid cell sharing either an edge or a vertex with him is empty or occupied by another module; in this last case, it can obtain information about the state of such module. No information about any other position is required.

Memory and computation The application of this set of rules requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to check if preconditions of rules are fulfilled, to memorize fixed values in some counters, and to make a simple operation with some counter at each step.

Number of moves As we have already analyzed the case of free locomotion, we focus here on the maximal number of moves performed by a module from the moment of its activation on the left of the system, to its inactivation on the right. In the following h will be the initial height of the rectangle and n the total number of modules.

In Figure 5.8 we can observe how, due to the presence of the obstacle, a system initially configured as a rectangle reaches the worst possible shape for the number of moves for a module; in such configuration, in fact, the number of inactive modules over which the current one needs to walk two or more times is maximal: both sides of each column of height h are free, and the number of columns that reach the maximal height is maximal, given that a separation of at least three cells between them is required in order to avoid an obstruction of the way. In addition, the current module can reach all the rest of the system, so even the modules used as a separation between the maximal columns generate the maximal number of moves, and no bridge is generated during the locomotion.

While the system is configured as described, each active module that activates on the left need to walk twice (three times in the case of the topmost modules) over the modules forming the columns of height h, and once over the rest of the modules (twice for the rightmost module) before

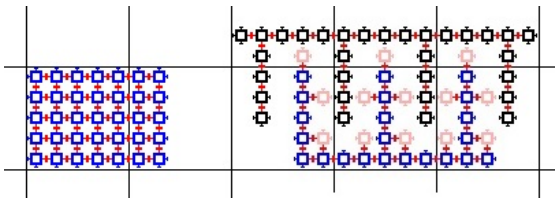


Figure 5.8: The rectangle reaches the worst configuration for the number of moves of a module while crawling under a superior obstacle; notice that the highest columns of the system under the superior obstacle are as high as the system in its initial configuration.

inactivating again; this means that in the worst case possible an active module performs $O(n)$ moves from the moment of its activation until it inactivates again. Indicating as k the number of rounds performed by the module during the locomotion of the system, a module performs at most $O(kn)$ moves, where k is directly proportional to the number of modules of the obstacle.

Communication A constant size communication is performed during the application of each rule, so the communication performed by each module is linear in the number of rules applied by the module.

Number of time steps The maximal number of time steps needed for the overpassing of a superior obstacle can be deduced by the examination of the best possible case, depicted in Figure 5.9: in this configuration, each time a module moves it performs a movement in the east direction, contributing to an overall movement towards east; in the case of the free locomotion, on the contrary, for each round, modules had to climb the system moving north, walk over the system and then reach their position moving south, exploiting time steps for the reconfiguration of the columns. We can conclude that the more the columns divide themselves under the

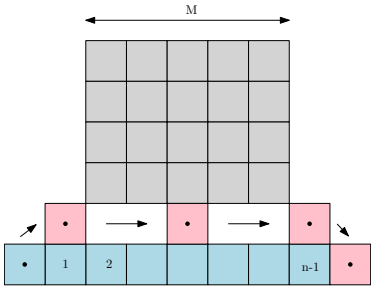


Figure 5.9: Best case for the number of time steps: the current module walks over all the other modules, and then deactivates again on the right. The current module is the module depicted in pink, and the inactive modules are blue; the obstacle modules are gray.

obstacle, the faster is the overpassing of the obstacle; the worst case for

the number of time steps is then the case in which the distance between the obstacle and the ground is big, and the system moves reconfiguring its columns completely as in the free locomotion; the overall number of time steps is then $O(kn)$ in the worst case, where k is the number of rounds of the system which is directly proportional to the width of the obstacle.

In the best case, with an obstacle of width M and a system with n modules always configured as a worm during the overpassing, the number of time steps needed for the overpassing is the number of steps from the moment in which the first module of the system is under the first column of the obstacle to the moment in which the last module of the system is under the last column of the obstacle; as a constant delay of 3 time steps is produced between the inactivation of a module and the inactivation of the following one in the worm, the overall number of time steps is O(3(M+n)) in the best case.

5.8 Rules

The algorithm is based on 12 different rules; before starting the locomotion each module has the following information stored in its counters: the vertical heigh of the module (C00), the initial heigh of the system (C01), the horizontal position of the module (C03) and the number of rounds (C04).

Rule name: North Priority: 4 Preconditions:!SOBSTA !SSTOPP N0*1* T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV Postconditions: P0,1 SACTIV A**1* C000 + C000 0001

- **Rule**: North
- **Priority**: 4.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N0*1*: concerns only the modules without a north neighbor and with a east neighbor.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A**1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the value of the counter C01 is increased by 1.

Rule name: North-east Priority: 4 Preconditions:!SOBSTA !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV Postconditions: P1,1 SACTIV A**11 C000 + C000 0001 C003 + C003 0001

- **Rule**: North-east
- **Priority**: 4.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N0*1*: concerns only the modules without a North neighbor and with a east neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C00 is increased by 1.
C003 + C003 0001: the counter C03 is increased by 1.

Rule name: East
Priority: 5
Preconditions:!SOBSTA !SSTOPP N**01 !T1,-1,ACTIV !T1,-1,OBSTA !E1,-1 !T0,1,INACT !T0,1,ACTIV
Postconditions: P1,0 SACTIV A0**1 C003 + C003 0001

- **Rule**: East
- **Priority**: 5.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N**01: the rule concerns only the modules without a west neighbors, but with a south neighbor
T1,-1,INACT: the support module needs to exist and to be set on *inactive*.
T1,-1,OBSTA: the support module needs to exist and to be set on *obstacle*.
!E1,-1: the grid cell in position $(1, -1)$ cannot be empty.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T0,1,INACT: the module in position $(0, 1)$ cannot be set on *inactive*.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to active.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C003 + C003 0001: the counter C03 is increased by 1.

Rule name: South-east
Priority: 4
Preconditions: !SOBSTA !SSTOPP N*001 E1,-1 !T0,1,INACT !T0,1,ACTIV !T-1,-1,ACTIV
Postconditions: P1,-1 SACTIV A*1*1 C000 - C000 0001 C003 + C003 0001

- **Rule**: South-east
- **Priority**: 4.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N*001: concerns the modules with a south neighbor and without a west nor a east neighbor.
E1,-1: the goal position needs to be empty.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T0,1,INACT: the module in position $(0, 1)$ cannot be set on *inactive*.
!T-1,-1,ACTIV: the module in position $(-1, -1)$ cannot be set on *active*.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1*1: the module attaches to its new west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the value of the counter C00 is decreased by 1. C003 + C003 0001: the counter C03 is increased by 1.

Rule name: South-east II
Priority: 5
Preconditions: !SOBSTA !SSTOPP N0*01 T1,1,OBSTA E1,-1
Postconditions: P1,-1 SACTIV A0101 C000 - C000 0001 C003 + C003 0001

- **Rule**: South-east II
- **Priority**: 5.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N0*01: concerns the modules with a south neighbor and without a north nor a east neighbor.
E1,-1: the goal position needs to be empty.
T1,1,OBSTA: the module in position $(1, 1)$ needs to be set on *obstacle*.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A0101: the module attaches to its new west neighbor but it does not attach to its new north and east neighbor; if it was attached before and if still possible, it attaches to the other neighbor.
C000 - C000 0001: the value of the counter C00 is decreased by 1. C003 + C003 0001: the counter C03 is increased by 1.

Rule name: South
Priority: 4
Preconditions: !SOBSTA !SSTOPP N01*0 T-1,-1,INACT !T1,0,INACT !T1,0,ACTIV
Postconditions: P0,-1 SACTIV A**11 C000 - C000 0001

- **Rule**: South
- **Priority**: 4.
- **Preconditions**:
!SSTOPP !SOBSTA: the rule is not applicable to modules set in the *stop* or in the *obstacle* state.
N01*0: concerns the modules with a west neighbor, without a module on the north and on the south.
T-1,-1,INACT: the support module needs to exist and to be set on *inactive*.
!T1,0,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T1,0,INACT: the module in position $(0, 1)$ cannot be set on *inactive*.
- **Postconditions**:
P0,-1: the module moves south.
A**11: the module attaches to its new south neighbor, if present; if it was attached before and where still possible, it attaches to the other neighbors.
C000 - C000 0001: the value of the counter C00 is decreased by 1.

Rule name: Deactivation
Priority: 1
Preconditions: SACTIV N*10* !T0,1,OBSTA
Postconditions: SINACT

- **Rule**: Deactivation
- **Priority**: 1
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.

N*10*: concerns the modules with a west neighbor and without a module on the east. The presence of a north and a south neighbor is not important.
!T0,1,OBSTA: the module in position $(0, 1)$ cannot be set on the *obstacle* state.
- Postconditions:
SINACT: the module changes its state to *inactive*.

Rule name: Deactivation II Priority: 6 Preconditions: SACTIV N0001 !T-1,1,OBSTA !T1,1,OBSTA E1,-1 < C000 C001 Postconditions: SINACT

- Rule: Deactivation II
- Priority: 6
- Preconditions:
SACTIV: it is applicable only to modules set in the *active* state.
N0001: concerns the modules with a south neighbor and without any other neighbor.
!T-1,1,OBSTA: the module in position $(-1, 1)$ cannot be set on the *obstacle* state.
!T1,1,OBSTA: the module in position $(1, 1)$ cannot be set on the *obstacle* state.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
! >C000 C001: the value of the counter C00 has to be bigger than the value of C01.
- Postconditions:
SINACT: the module changes its state to *inactive*.

Rule name: Stop Priority: 10 Preconditions: SINACT N0011 > C003 C004 = C000 C001 Postconditions: SSTOPP

- Rule: Stop
- Priority: 10.
- Preconditions:
SINACT: it is applicable only to modules set in the *inactive* state.
N0011: concerns the modules with a west and a south neighbor, and without a module on the north and on the west.
= C000 C001: the value of the counter C00 needs to meet the value of the counter C01
> C003 C004: the value of the counter C03 needs to be bigger than the value of the counter C04
- Postconditions:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop II
Priority: 10
Preconditions: SINACT T-1,0,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 2
- **Priority**: 10.
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
T-1,0,STOPP: the east neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop III
Priority: 10
Preconditions: SINACT T-1,-1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 3
- **Priority**: 10.
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
T-1,1,STOPP: the grid cell with relative coordinates $(-1, 1)$ needs to be occupied by a module set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop IV
Priority: 10
Preconditions: SINACT T0,1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop 4
- **Priority**: 10.
- **Preconditions**:
SINACT: it is applicable only to modules set on the *inactive* state.
T0,1,STOPP: the north neighbor needs to be set in the *stop* state.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Chapter 6

Tunneling of a rectangle

6.1 Goal

In this chapter we discuss the eastward locomotion of a rectangular system in the presence of both superior and inferior obstacles (i.e. both low and high obstacles); we refer to this kind of situation as 'tunneling', as superior and inferior obstacles can combine together to form tunnels. In our setting, obstacles are configured as histograms and are not subject to height restrictions; the minimum vertical distance between a superior obstacle and the floor, and between a superior and an inferior obstacle, is of two grid cells; moreover, the first neighbor of a superior obstacle module cannot overlap with the one of an inferior obstacle module, i.e. there has to be at minimum a 2-grid-cells diagonal distance between the two types of obstacles. As will be seen, this restriction is necessary in order to guarantee the connection of the system.

6.2 Strategy

The set of rule presented is the union of the rules of the rectangle locomotion under superior obstacles and the rules of the locomotion in presence of high obstacles, with some adaptations and changes that make the rules work together properly. New states are assigned to the different types of obstacle, in order to make the system able to decide how to behave in the different cases; the *obsth* and the *obstl* states are assigned respectively to the superior and the inferior obstacles. During the locomotion, when the system encounters an inferior obstacle, it overpasses it following the strategy of the rules for general histogram obstacles presented in Chapter 4; in the presence of superior obstacles, and in the case in which the two kinds of obstacles are combined forming a tunnel, the system continues its locomotion just limiting its height when necessary, following the strategy of the locomotion under superior obstacles presented in Chapter 5. Due to the distance conditions between the obstacles, each module of the system deals separately with superior and inferior obstacle modules without any interference between the two different strategies of movement. We can see an example of the tunneling of a rectangle in Figure 6.1.

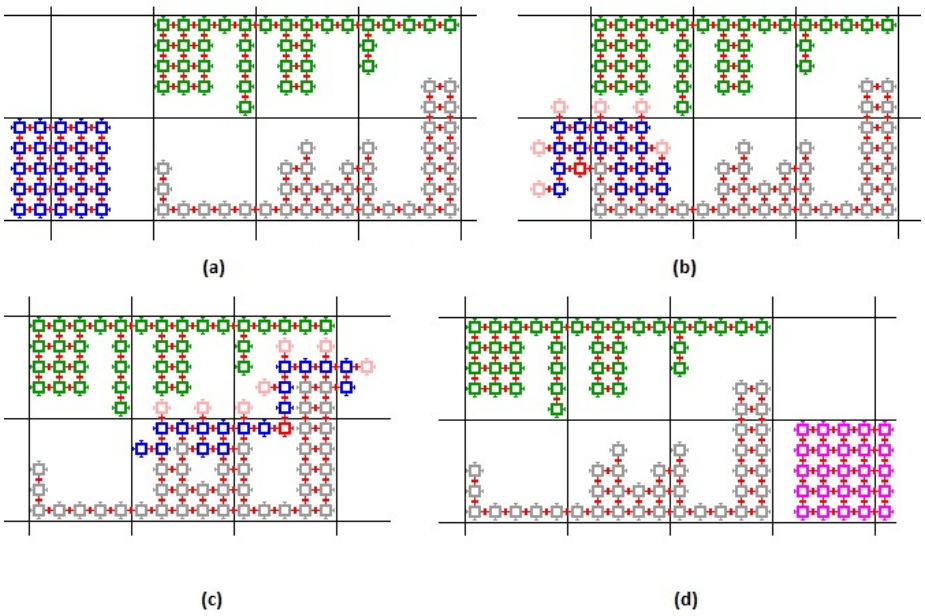


Figure 6.1: Tunneling of a rectangular system: the different colors for the superior and the inferior obstacles indicate the two different states assigned; in (a) we can see that the system is configured as a rectangle before the start of the locomotion; in (b) and (c) the overpassing of the tunnel is performed, maintaining the initial height in the low parts of the obstacle, and configuring as a worm in the high parts. In (d), the system reconfigures after the overpassing.

6.3 Correctness

As the strategy is only a combination of the different approaches explained in the previous chapters, we skip here the proofs of the correctness of the rules, and we present directly the detailed rules in the following section. Notice that the fact that the first neighborhood of a superior obstacle module cannot overlap with the one of an inferior obstacle module assures that there are no compatibility problems between the rules; no module can find itself in a situation in which the rules for the overpassing of an inferior obstacle are not applicable because of the presence of the superior obstacle, or viceversa, if this distance restriction is respected. Moreover, the priorities of the rules are reestablished in order to avoid the application of two or more rules at a time, so each module can still apply at most one rule at each time steps.

6.4 Complexity

Neighbourhood During the locomotion each module is able to check if any grid cell sharing either an edge or a vertex with it is empty or occupied by another module; the in this last case, it can obtain information about the

state of such module. In some situations the same information is needed for some of the grid cells of the second neighbor; such extension is restricted to the case in which the stability of the system could be compromised, and to prevent disconnection in the case of a 2 columns rectangle.

Memory and computation The application of this set of rules only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to memorize a fixed value in a counter, to perform some simple operations with counters, and to check if preconditions of rules are fulfilled.

Number of moves For the locomotion of a rectangle in the presence of high obstacles, the worst case possible for the number of moves of a module from its activation to its first following inactivation is $O(n)$, reached in a configuration in which the module needs to walk over all the rest of modules before inactivating again. Such case cannot be worsen by the presence of superior obstacles over the system, as the only effect superior obstacles have is the flattening of the system, which in the worse case is already flat (i.e. configured as a worm). The total maximum number of moves for a module is then $O(kn)$, where k is the number of rounds performed by the system during the locomotion, and is directly proportional to the number M of the modules of the inferior obstacle.

Communication A constant size communication is performed during the application of each rule, so the communication performed by each module is linear in the number of moves.

Number of time steps As we have already explained, the presence of superior obstacles cannot increase the number of time steps of the locomotion, as the more the columns divides themselves under the obstacle, the faster is the overpassing of the obstacle; the worst case for the number of time steps for the tunneling is then the case in which the distance between the superiors obstacles and the high obstacles is big, and superior obstacles are not perceived by the system; the number of time steps is then at most the one computed for the case of high obstacles: $O(kn)$, where n is the total number of time steps, and k is the number of rounds of the system, which is directly proportional to the number of modules M of the inferior obstacle.

6.5 Rules

The algorithm is based on 23 different rules; before starting the locomotion each module has the following information stored in its counters: the vertical heigh of the module (C00), the initial heigh of the system (C02), the nuber of columns of the system in its initial configuration (C03) and the number of rounds (C05).

Rule name: North I
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*10 T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV
Postconditions: P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North I
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*10: the rule concerns the module without a north neighbor and with a east neighbor.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North II
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*11 !E1,-1 T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV !(T1,-1,OBSTL !T0,-1,OBSTL)
Postconditions: P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North II
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*11: the rule concerns the module without a north neighbor and with a east and a south neighbor.
!E1,-1: the grid cell in position $(1, -1)$ needs to be occupied.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!(T1,-1,OBSTL !T0,-1,OBSTL): the rule is not applicable when the module in position $(1, -1)$ is an inferior obstacle and the module in $(0, -1)$ is not an inferior obstacle.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.

A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North III
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*11 E1,-1 T1,1,INACT !T-1,0,INACT !T-1,0,ACTIV T0,-1,OBSTL
Postconditions: P0,1 SACTIV A1*1* C000 + C000 0001

- **Rule**: North III
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*11: the rule concerns the module without a north neighbor and with a east and a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
T1,1,INACT: the support module needs to exists and to be set on *inactive*.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
T0,-1,OBSTL: the module in position $(0, -1)$ needs to be set on *obstl*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches afterward if was attached before, when possible.
C000 + C000 0001: the counter C00 is increased by one.

Rule name: North-east
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*1* T1,0,INACT E1,1 !T-1,0,INACT !T-1,0,ACTIV ! (= C003 0002 T2,1,OBSTH E2,0)
Postconditions: P1,1 SACTIV A***1 C000 + C000 0001 C004 + C004 0001

- **Rule**: North-east
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*1*: concerns only the modules without a North neighbor and with a east neighbor.
T1,0,INACT: the east neighbor needs to be set on *inactive*.
E1,1: the goal grid cell needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
!(= C003 0002 T2,1,OBSTH E2,0): the rule cannot be applied if the value of the counter C03 is 2, the module in position (2,1) is set on *obsth* and the grid cell in position (2,0) is empty.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

C004 + C004 0001: the counter C004 is increased by 1 each time the rule is applied.

Rule name: East
Priority: 5
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N**01 !T1,-1,ACTIV !T1,-1,OBSTL !E1,-1 !T0,1,INACT !T0,1,ACTIV !T0,-1,OBSTH !T0,-1,OBSTL
Postconditions: P1,0 SACTIV A**11

- **Rule**: East
- **Priority**: 5
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N**01: the rule concerns only the modules without a west neighbors, but with a south neighbor.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,-1,OBSTL: the module in position $(1, -1)$ cannot be set on *obstacle*.
!E1,-1: the grid cell in position $(1, -1)$ cannot be empty.
!T0,-1,OBSTL: the module in position $(0, -1)$ cannot be set on *obstl*.
!T0,-1,OBSTH: the module in position $(0, -1)$ cannot be an *obsth*.
!T0,1,INACT: the module in position $(0, 1)$ cannot be set on *inactive*.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule name: South
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N*1*0 T-1,0,INACT T-1,-1,INACT !T1,0,INACT !T1,0,ACTIV
Postconditions: P0,-1 SACTIV A**11 C000 - C000 0001

- **Rule**: South
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N*1*0: concerns the modules with a west neighbor, without a module on the south.
!T1,0,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T-1,0,INACT: the module in position $(-1, 0)$ needs to be set on *inactive*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new west and south neighbors; if it was attached before and where still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east I
Priority: 4
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N*001 E1,-1 !T0,1,INACT !T0,1,ACTIV !T-1,-1,ACTIV
Postconditions: P1,-1 SACTIV A*1*1 C000 - C000 0001

- **Rule**: South-east I
- **Priority**: 4
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N*001: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
!T0,1,INACT: the module in position $(0, 1)$ cannot be set on *inactive*.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
!T-1,-1,ACTIV: the module in position $(-1, -1)$ cannot be set on *active*.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*1*1: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east II
Priority: 10
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*01 T1,1,OBSTH E1,-1
Postconditions: P1,-1 SACTIV A010* C000 - C000 0000

- **Rule**: South-east II
- **Priority**: 10
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*01: the rule concerns only the modules without north or west neighbors, but with a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
T1,1,OBSTH: the grid cell in position $(1, 1)$ needs to be set to *obsth*.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A0101: the module attaches to its new west and south neighbor; it doesn't attach to its north neighbor and east neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east III
Priority: 6
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*01 E1,-1 !T-1,0,ACTIV !T-1,0,INACT > C000 C002
Postconditions: P1,-1 SACTIV A*1*1 C000 - C000 0001

- **Rule**: South-east III
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*01: the rule concerns only the modules without north or east neighbors, but with a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
!T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
!T-1,0,ACTIV: the module in position $(-1, 0)$ cannot be set on *active*.
> C000 C002: the value of the counter C00 needs to be bigger than the value of C02.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east IV
Priority: 5
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0*01 E1,-1 !T0,-1,OBSTL !T0,-1,OBSTH > C000 C002
Postconditions: P1,-1 SACTIV A*11* C000 - C000 0001

- **Rule**: South-east IV
- **Priority**: 5
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0*01: the rule concerns only the modules without north or east neighbors, but with a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
!T0,-1,OBSTL: the module in position $(0, -1)$ cannot be set on *obstl*.
!T0,-1,OBSTH: the module in position $(0, -1)$ cannot be an *obsth*.
> C000 C002: the value of the counter C00 needs to be bigger than the value of C02.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: South-east V
Priority: 9
Preconditions: !SOBSTH !SOBSTL !SBRIDG !SSTOPP N0101 E1,-1 E0,-2 !(!T-1,-2,OBSTH !T-1,-2,OBSTL)
Postconditions: P1,-1 SACTIV A*11* C000 - C000 0001

- **Rule**: South-east IV
- **Priority**: 9
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N0101: the rule concerns only the modules without north or east neighbors, but with a west and a south neighbor.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
E0,-2: the grid cell in position $(0, -2)$ needs to be empty.
!(!T-1,-2,OBSTL !T-1,-2,OBSTH): the module in position $(1, -2)$ has to be set either on *obstl* or *obsth*.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: North-west
Priority: 3
Preconditions: !SOBSTH !SOBSTL !SSTOPP N10** E-1,1 !(!T1,0,OBSTL !T1,0,OBSTH) !T0,-1,INACT !T0,-1,BRIDG !T0,-1,ACTIV
Postconditions: P-1,1 SACTIV A*11* C000 + C000 0001

[h]
- **Rule**: North-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH: the rule is not applicable to modules set in the *stop*, *obstl* or *obsth* state.
N10**: the rule concerns only the modules with a north neighbor, but without a west neighbor.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T0,-1,INACT: the module in position $(0, -1)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *active*.
!(!T1,0,OBSTL !T1,0,OBSTH): the module in position $(1, 0)$ has to be set either on *obstl* or *obsth*.
- **Postconditions**:
P-1,1: the module moves north-west.
SACTIV: the module changes its state to *active*.
A*11*: the module attaches to its new west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: North-west II
Priority: 3
Preconditions: !SOBSTH !SOBSTL !SSTOPP !SBRIDG N100* !T1,1,ACTIV !T1,1,INACT !T1,1,OBSTH !T1,1,OBSTL E-1,1 !T0,-1,INACT !T0,-1,ACTIV !T0,-1,BRIDG
Postconditions: P-1,1 SACTIV A**1* C000 + C000 0001

- **Rule**: North-west II
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N100*: the rule concerns only the modules with a north neighbor, but without a west or east neighbor.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T1,1,INACT: the module in position $(1, 1)$ cannot be set on *inactive*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T1,1,OBSTL: the module in position $(1, 1)$ cannot be set on *obstl*.
!T1,1,OBSTH: the module in position $(1, 1)$ cannot be set on *obsth*.
!T0,-1,INACT: the module in position $(0, -1)$ cannot be set on *inactive*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *active*.
- **Postconditions**:
P-1,1: the module moves north-west.
SACTIV: the module changes its state to *active*.
A**1*: the module attaches to its new west neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C000 + C000 0001: the counter C000 is increased by 1 each time the rule is applied.

Rule name: South-west
Priority: 3
Preconditions: !SOBSTH !SOBSTL !SSTOPP !SBRIDG N*1*0 !(!T-2,-1,OBSTL !T-2,-1,OBSTH) !T-1,0,OBSTL !T-1,0,OBSTH E-1,-1 T-1,0,INACT
Postconditions: P-1,-1 SACTIV A11** C000 - C000 0001

- **Rule**: South-west
- **Priority**: 3
- **Preconditions**:
!SSTOPP !SOBSTL !SOBSTH !SBRIDG: the rule is not applicable to modules set in the *stop*, *bridge*, *obstl* or *obsth* state.
N*1*0: the rule concerns only the modules without a south neighbor, but with a west neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
T-1,0,INACT: the module in position $(-1, 0)$ needs to be set on *inactive*.
!T-1,0,OBSTL: the module in position $(-1, 0)$ cannot be set on *obstl*.
!T-1,0,OBSTH: the module in position $(-1, 0)$ cannot be set on *obsth*.
!(!T-2,-1,OBSTL !T-2,-1,OBSTH): the module in position $(-2, -1)$ needs to be set either on *obstl* or on *obsth*.
- **Postconditions**:
P-1,-1: the module moves south-west.
SACTIV: the module changes its state to *active*.
A11**: the module attaches to its new north and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.

C000 - C000 0001: the counter C000 is decreased by 1 each time the rule is applied.

Rule name: Formation of bridges
Priority: 6
Preconditions: SACTIV N0110 T1,0,OBSTL != C000 0001
Postconditions: SBRIDG

- **Rule**: Formation of bridges
- **Priority**: 6
- **Preconditions**:
SACTIV: the module changes its state to *active*.
N0110: the rule concerns only the modules without a north and south neighbor, but with a west and a east one.
T1,0,OBSTL: the module in position $(1, 0)$ needs to be set on *obstl*.
!C000 0001: the value of the counter C000 needs to be different than 1.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.

Rule name: Deactivation I
Priority: 2
Preconditions: SACTIV N*10* !T0,1,OBSTH !T0,-1,ACTIV
Postconditions: SINACT

- **Rule**: Deactivation I
- **Priority**: 2
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N*10*: the rule concerns only the modules with a west neighbor.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T0,1,OBSTH: the module in position $(0, 1)$ cannot be set on *obsth*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Deactivation II
Priority: 7
Preconditions: SACTIV N0001 !T-1,1,OBSTL !T-1,1,OBSTH !T1,1,OBSTL !T1,1,OBSTH !T0,1,OBSTL !T0,1,OBSTH
E1,-1 !> C000 C002
Postconditions: SINACT

- **Rule**: Deactivation II
- **Priority**: 7
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N0001: the rule concerns only the modules without a north, east or west neighbor, but with a south neighbor.
!T-1,1,OBSTL: the module in position $(-1, 1)$ cannot be set on *obstl*.
!T-1,1,OBSTH: the module in position $(-1, 1)$ cannot be set on *obsth*.
!T1,1,OBSTL: the module in position $(1, 1)$ cannot be set on *obstl*.
!T1,1,OBSTH: the module in position $(1, 1)$ cannot be set on *obsth*.
!T0,1,OBSTL: the module in position $(0, 1)$ cannot be set on *obsth*.
!T0,1,OBSTL: the module in position $(0, 1)$ cannot be set on *obsth*.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.

!>C000 C002: the value of the counter C00 cannot be bigger than the value of C02.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Deactivation III Priority: 1 Preconditions: SACTIV N0*** !T1,0,INACT !T1,0,ACTIV Postconditions: SINACT

- **Rule**: Deactivation III
- **Priority**: 1
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N0***: the rule concerns only the modules without a north neighbor.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T1,0,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Deactivation IV Priority: 1 Preconditions: SACTIV T-1,0,INACT T1,0,INACT Postconditions: SINACT

- **Rule**: Deactivation IV
- **Priority**: 1
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *inactive* state.
N0***: the rule concerns only the modules without a north neighbor.
T-1,0,INACT: the module in position $(-1, 0)$ cannot be set on *inactive*.
T1,0,INACT: the module in position $(1, 0)$ cannot be set on *inactive*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule name: Stop I Priority: 10 Preconditions: SINACT N0011 = C004 C005 = C000 C002 Postconditions: SSTOPP

- **Rule**: Stop I
- **Priority**: 10
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N0011: the rule concerns only the modules without a north and east neighbor, but with a west and a south one.
= C004 C005: counter C04 needs to meet the value of C05.
= C000 C002: counter C00 needs to meet the value of C02.
- **Postconditions**:
SSTOPP: the module changes its own state to *stop*.

Rule name: Stop II
Priority: 10
Preconditions: SINACT T-1,0,STOPP
Postconditions: SSTOPP

- **Rule**: Stop II
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T-1,0,STOPP: the module in position $(-1, 0)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop III
Priority: 10
Preconditions: SINACT T0,1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop III
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T0,1,STOPP: the module in position $(0, 1)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Rule name: Stop IV
Priority: 10
Preconditions: SINACT T-1,-1,STOPP
Postconditions: SSTOPP

- **Rule**: Stop IV
- **Priority**: 10
- **Preconditions**:
SINACT: the module changes its state to *inactive*.
T-1,-1,STOPP: the module in position $(-1, -1)$ needs to be set on *stop*.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.

Chapter 7

Histogram locomotion

7.1 Goal

The purpose of the set of rules presented in this chapter is to produce the eastward locomotion of any modular robotic system initially configured as a connected histogram. The locomotion is performed on a free plane ground (a horizontal line) without obstacles.

7.2 Locomotion strategy

The rules produce the locomotion by making the modules of the leftmost column in turn move from the back of the group, over the top, and locate on the front of the group to reform a new column with the same height; the height and the relative position of the columns are invariant during the whole locomotion. All modules are initially state to *stop*; the modules of the leftmost column of the system are the first to move; as a module happens to be the bottommost of the leftmost column, it changes its state from *stop* to *inactive*; the other modules of the column do the same, until the topmost one changes its state to *inactive* and then activates, free to apply an action rule. As the topmost module moves, the others of its column activate and move, walking over the stopped modules until they reach the right of the system. As the second column becomes the leftmost, it is free to move in the same way.

During the locomotion some bridges are created in order to avoid filling the bottlenecks of width 1 formed between the columns, as we can observe in Figure 7.1. Each module set on *bridge* allows other modules to pass by without having to fill all the spaces over the system.

A bridge activates again as the modules of the column on its left change their state to *inactive*.

In order to recreate a column with the right height and position, the order among the modules has to be preserved during the locomotion, and modules of the same column have to reach together the right part of the system; in order to achieve this, some information has to be stored in each module, such as the height of the column and its relative position, and this information has to be transmitted between the modules along the locomotion; this transmission is realized by the introduction of a new state: the *rinfo* state: as two modules

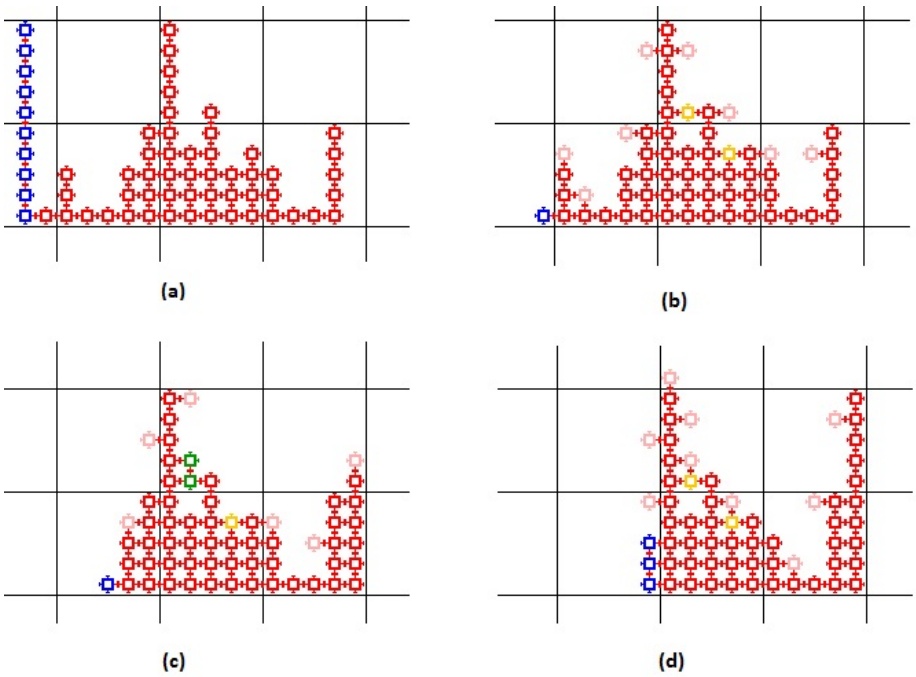


Figure 7.1: Histogram locomotion: in (a) the modules of the first column change their state from *stop* to *inactive*. In (b) they activate, starting the locomotion; in (c) and (d) we can observe that the same column is reformed on the right part of the system, with the same height. In (b), (c) and (d) the formation of bridges in the bottlenecks of width 1 is shown. Active modules are depicted in pink, inactive modules are blue, bridges are orange; the green modules are set in the *rinfo* state, exchanging information about the columns.

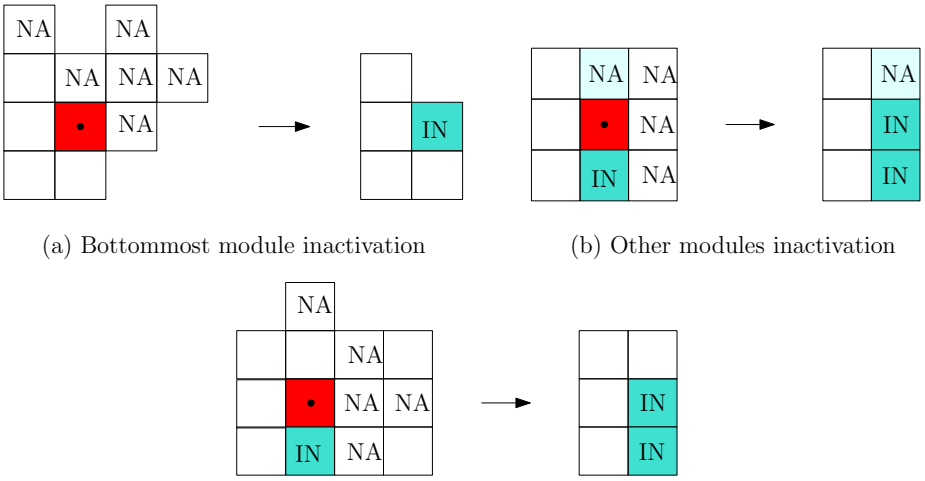
need to exchange some information they set to the *rinfo* state, communicate, and go back to the original state. The only moment in which the order between the modules is altered is when a module surpasses a bridge, so the exchange of information along the locomotion is always performed between a bridge and an active module. As the system always mantains the order and the height of the column during the locomotion, the reconfiguration of the system mostly consists in stopping the locomotion when the first column of the initial configuration of the histogram is the leftmost column.

7.3 Activation, locomotion and bridges

The initial configuration of the system is an histogram with all the modules set to the *stop* state.

The shape of the histogram is stored into the modules, codified through the counters $C01$ and $C02$. The counter $C01$ memorized in each module stores the value indicating the relative position of the column of the module: starting

from 1 for the first column on the left, the columns are numbered from the left to right. The counter $C02$ stores the height of its column, so that each module knows the number of modules of the column it belongs to. The change of state from *stop* to *inactive* is done through the application of one of the 3 rules depicted in Figure 7.2.



(a) Bottommost module inactivation

(b) Other modules inactivation

(c) Topmost module inactivation

Figure 7.2: Inactivation rules for the modules: the current module is indicated with a dot, and empty grid cells are indicated with an empty square. Colored grid cells are occupied cells, red cells are modules set in the *stop* state. NA indicates that if a module occupies that grid cell, it cannot be active, while IN indicates that it has to be inactive.

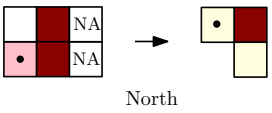
As can be noticed, these rules apply only to the modules of the leftmost column, changing their states from the bottommost to the topmost of the column. As soon as the topmost module changes its state to *inactive*, it is able to apply one of 5 action rules (North, North-East, South, South-East and East), activating and moving on the top of the system, leaving the other modules free to move. As an example of action rule, we can see Figure 7.3 which depicts the North rule. For the details of the other action rules, see Section 7.9.

Once all the modules of the first column have activated and moved, the second column becomes the first, and the process is repeated.

While walking over the other modules of the system, any active module that encounters a bottleneck of width 1 formed by two different columns applies the Bridge Rule depicted in Figure 7.4, changes its state to *bridge* and blocks the bottleneck in order to prevent obstruction.

Bridges may move during the locomotion of the system, as we can observe in Figure 7.5; the rule for the movement of bridges is depicted in Figure 7.6.

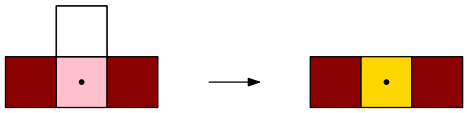
A module which is set on *bridge* activates and moves again as the modules of the column on its left are ready to move. The rules for the the activation and movement of a bridge are depicted in Figure 7.7



North

North 6 State≠(Stop,Still,Bridge)&(0,1)=empty&(1,0)=Full&State(0,2)≠Active&State(1,2)≠Active& State(1,1)=Stop Move (0,1) & State=Active& C007 = 0000 & C017 = 0000

Figure 7.3: Detailed example of the structure of one of the advance rules; notice that not all the preconditions of the rule are included in the graphic representation. The preconditions over the second neighborhood of modules are introduced to generate a delay between the active modules while they are moving, as the elimination of the gap between them may cause collisions between active modules and moving bridges.



Formation of a Bridge

Figure 7.4: Rule for the formation of a bridge.

7.4 Information exchange

As the active modules of one column walk over the system and reach the right part, they reconfigure through the application of two specific rules. The first rule stops the first active module that arrives at the bottommost position on the right of the system, and its column is reconfigured through the second rule. This process does not preserve the initial shape of the histogram if the order of the modules is not respected during the locomotion: if a module of the $i+1$-th column arrives before the modules of column i, column $i+1$ is reconfigured before the i-th column, and the logical order of the columns of the histogram is modified during the locomotion. In order to recreate the columns with the right position, the order among the modules has to be preserved; this is achieved through the exchange of information performed during the locomotion. As the only moment in which the order between the modules can be altered is when an active module of the column j surpasses a bridge which belongs to a column $i < j$, these rules apply only to this kind of situation.

The exchange between an active module and a bridge is performed in two phases: the first phase consists of the communication between the modules, controlled by the rules "Info exchange: active" and "Info exchange: bridge", which are two symmetric rules for the exchange of the counters, the first applicable only to the active modules on the top of the bridge, and the second applicable only to the bridge. If two modules fulfill the preconditions, they change their state to *rinfo*, read the values of the counters $C01$ and $C02$ of the other module and memorize them into two internal auxiliary counters $C011$ and $C012$.

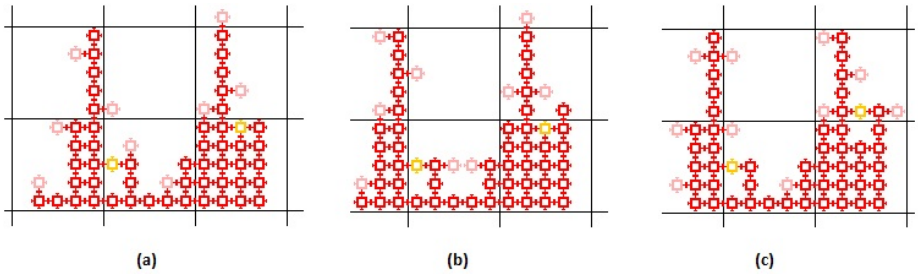


Figure 7.5: In (a) the bridge is formed between a complete column and the last column, still incomplete; as another module of the last column changes its state to *stop* and becomes the topmost one in (b), we can see as in (c) the bridge moves North, while the active modules wait in order to avoid collisions.

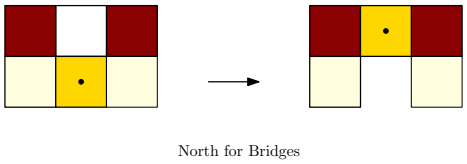


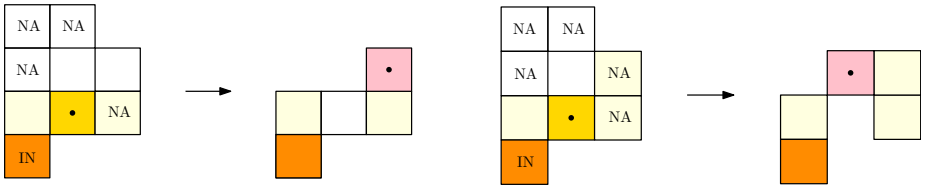
Figure 7.6: North rule for the bridges.

The second phase is an internal change of information: each module writes the values taken from the other into its own counters $C01$ and $C02$, erasing the original values, and goes back to its previous state. Through these rules then the two modules swap their counters, changing their role in the reconfiguration of the system, and avoiding the change of order of the columns. For the details of these rules, see Section 7.9.

7.5 The case of the last column

As explained in the previous section, the exchange of information is only performed when an active module of a column i walks over a bridge of a column j, where $j < i$. This condition cannot be used when the bridge is a module of the last column; in this case in fact, as column 1 activates and walks over the bridge, the exchange of information would not be performed, as the precondition $j < i$ is not fulfilled, and column 1 would be reformed before the last column, changing the original order of the columns.

This problem is solved by the creation of some rules which treat specifically the case of a bridge of the last column and an active module of column 1. These rules force the exchange of information between such modules and resolve the problem, as we can easily notice that there are no other situations in which the swap is not performed when it has to: in fact, a contact between modules of column that are not consecutive (in a circular order where the first column follows the last one) is always prevented by the passage of the modules of the



(a) North-East activation for bridges

(b) North activation for bridges

Figure 7.7: Rules for the activation and movement of bridges; notice that these bridges activates and move through the same rule.

intermediate columns; the last column interacts then only with the first and the penultimate column. For the detailed rules see Section 7.9.

7.6 Reconfiguration

As the system always preserves the order and the height of the columns during the locomotion, the idea of the reconfiguration is to stop the locomotion when column 1 is again the leftmost column of the system.

In order to achieve this, some other counters are stored in the memory of the modules. The counter C10 is stored into the memory of each module, fixed before the locomotion starts, and controls the number of rounds that the system need to perform before the locomotion stops; each time a module changes its state from *stop* to *inactive* a counter C04 is increased by one, counting its number of inactivations. When column 1 is again the leftmost column, and the counter C04 of its bottommost module equals the required value C10, such module changes its state to *still*, blocking the process of inactivation of its column. In order to complete the final reconfiguration of the system, some additional rules are needed. As we know, some bridges are formed during the locomotion, and they only activate again as the first of the two columns they connect change its state to inactive and is ready to move; in the final phase though, as we stop the first module and block the inactivation of the modules, the bridges cannot apply the activation rules of Figure 7.7 and need some additional rules to understand how and when they can activate and move.

In order to solve this problem, a counting phase is introduced: as soon as a module of the last column walks on the top of the bridge for the first time, the bridge becomes an element of the last column through a swap of information; then, the bridge starts the counting of the number of modules of the last column while they walk over it, and reactivate as the last module of the column has passed.

To perform this computation, some other counters are introduced. The process of the reactivation of a bridge starts when the counter C04 of the rounds has reached the fixed value C10. As soon as the value of the counter C01 of a bridge equals the counter C13, indicating the number of columns of the histogram (i.e. when the bridge becomes a module of the last column), it starts the counting of the modules of the last column while they pass over, and it sums

1 to C14 for each module. When C14 reaches the value of C15, indicating the number of modules of the last column, the bridge changes its state to *moves*, and is free to activate through the action rules.

As these bridges reach the right side of the system, they stop again, recreating the last missing columns, and the reconfiguration is complete.

7.7 Correctness

In this section we prove that the described rules actually produce an eastward locomotion of a robotic system initially configured as a connected histogram. In particular, we prove that:

1. No collisions are produced during the movement.
2. The system stays connected during the whole movement.
3. There is always at least one rule that is applicable to some module of the system.
4. The rules actually produce a locomotion of the system from West to East on free ground.
5. The system preserves the order and the height of the columns of the initial configuration while the locomotion is performed.

In order to prove these results, we firstly state and prove a preliminary result:

Lemma 5 *Each module can apply at most one action rule at a time.*

Proof: We need to check that rules with the same priority are pairwise incompatible, i.e. that a module cannot satisfy the preconditions of two of them at the same time. This is easily seen by analyzing the preconditions. Without loss of generality, we can examine the North and the North-East rules: a requirement for the application of North is that the cell grid with relative coordinates (1,1) is occupied by a module in the Stop state, while North-East requires the same cell to be empty in order to change the position, so no module can satisfy both sets of preconditions at the same time.

Each time two rules have the same priority we find similar contradictions, so the rules are pairwise incompatible and modules can apply only one rule at a time. □

Proposition 14 *No collisions are produced during the movement.*

Proof: The first possible type of collision is the one in which two different modules move to the same grid cell at the same time while applying the rules.

Due to Lemma 5 we only need to worry about conflicts created by the application of one action rule at a time. While in the case of the rectangle collisions where impossible due to the shape of the system and the nature of the movements, in this case there are many situations to study; in order to avoid conflicts, some specific preconditions over the second neighborhood of modules are introduced; such conditions exclude any possible collision between modules. For a better understanding of this, we can examine the South rule, described

in Figure 7.8. As a module applies the South rule, different types of collision can occur; the only module that can cause them is the one occupying the relative position $(0, -2)$, as we can easily notice that no other module would be able to move to the relative position $(0, -1)$ in such configuration. The cases in which the module in $(0, -2)$ is *active* or in the *moves* state, and able apply the locomotion rules, is explicitly excluded with the precondition 'State(0,-2) $\neq$(Active, Moves)'. The only other possible case is the one in which $(0, -2)$ is a bridge, willing to move to $(0, -1)$ either because it is reactivating, either because it is applying the North rule for bridges. The first case is not considered in the South rule, as it excluded in the Rules of activation of bridges through the precondition 'State(0,2) $\neq$ active'. The second case is contemplated in the South rule through the precondition 'Not(State(0,-2)=Bridge&State(1,-1)=Stop', which excludes this possibility too.

Move South
6

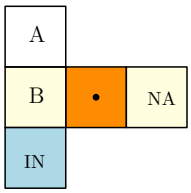
State≠(Stop,Still,Bridge)&(1,0)=(0,-1)=empty&(-1,0)=(-1,-1)=Full&State(0,-2)≠(Active,Moves,Rinfo)&State(1,-2)≠Rinfo&
Not(State(1,-2)=Active&State(0,-2)=Stop)&Not(State(0,-2)=Bridge&State(1,-1)=Stop)

Move (0,-1) & State=Active& C007 = 0000 & C017 = 0000

Figure 7.8: South rule

The second possible type of collision is the case of a module moving to a cell which is being used as an intermediate position for the movement of another module. To prove that a disconnection cannot occur in this case, we analyze the case of a bridge applying the North-east activation rule; all the other cases can be treated in the same way.

Let's suppose that a bridge activates moving North-east; observing the preconditions of the North-east rule, we can see that there are two different possibilities: any module moving to the relative position $(0, 1)$ has to start the movement from positions A or B of Figure 7.9.



i-column

Figure 7.9: A module moves from position A or B and tries to attach to the bridge, while the bridge activates through the North-east activation rule.

As one of the precondition of the North-east rule is that the support module is in the *stop* state, we can immediately discard the case in which a module in position B wants to move over the bridge applying the North-east rule.

Let's suppose now that a module in position A wants to move east. The module cannot be a stopped module, as each module has to pass through the

inactive state before moving; the cases of a *moves* module or a *bridge* are excluded by the configuration of the system, and a *still* module would not be applying an action rule in this case.

The only possibilities left for a module in A applying the East rule are the active and the inactive state. An *active* module in position A either is the topmost module of column i, either is an active module coming from a previous activated column. The first case is impossible as the bridge would have activated before the coming of any active module in position A. The second case can be excluded too: in the previous step, module A would have be in relative position $(-2, 0)$, and the module in position $(-1, -1)$ could not be inactive yet, because of the preconditions of the inactivation rules. The only case left is the case in which module A is an inactive module; in this case the whole column would be inactive and, as before, the bridge would have been already active. The other cases of collision with an activating bridge can be analyzed in an analogous way. As all the rules present these type of preconditions, collisions are always avoided.

□

Proposition 15 *The system stays connected during the whole movement.*

Proof: The histogram locomotion, as we have explained, consists in the locomotion of some active modules of the first columns walking over a static connected histogram formed by the other columns of the system; as these left columns do not perform any movement, and are always connected at least by their bottommost modules, the only modules that can generate a disconnection are the active modules while they walk over the system. Due to Lemma 5, the only cases to study are the ones originated by the application of one rule at a time. All the post-conditions of the action rules guarantee that the current module is connected to one of its neighbors after the movement, the 'support module' of the rule; the only situation in which a disconnection could occur is the one in which while a module applies a rule and intends to connect to the support module, this last one applies an action rule itself, leaving its grid cell empty. This is possible only if the support module is not set in the *stop* state, i.e. if it is not part of the static part of the histogram.

We can assume that before the current module applies the rule the system is connected, as in the initial configuration the system is connected. We will see now that the cases in which the support module is inactive cannot generate disconnections; all the other cases can be treated in an analogous way.

Firstly we can notice that the only inactive modules of the system belong to the leftmost column: modules of column i change their state from *Stop* to *inactive* only when column $i-1$ is already active, i.e. when column i is the leftmost column of the system. Moreover, the only inactive module able to move is the topmost module, so we can immediately exclude the case of an inactive module in the middle of the column disconnecting the system by the application of some action rule.

A stopped module changes its state to *inactive* only in the case in which the grid cells of the left half of its neighborhood are empty; the inactivation starts from the bottom to the top for each column; this means that the first column is inactive and ready to move only when all the active modules of the previous column have reached its right side. These observations exclude the case of an

inactive support for any module moving on the right side of the column (such as the North rule, the North-East rule, the East rule).

The only case of an inactive support for a module moving on the left side of the first column is the case depicted in Figure 7.10; we can notice that in this case the support module cannot apply any of the action rules, and then cannot generate any disconnection of the system.

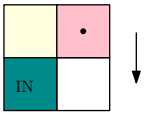


Figure 7.10: The moving module applies the South rule moving on an inactive module (blue in the picture): the support module cannot apply any of the action rules, as it does not fulfill any of the preconditions, so no disconnection can be generated.

□

Proposition 16 *There is always at least one rule that is applicable to some module of the system.*

Proof: The proof is analogous to the case of the free locomotion of the rectangle; we can analyze the movement of the leftmost column and proceed by induction.

Among the modules of the leftmost column there is always a module able to apply a rule, either an inactivation or an action rule. As the topmost is inactive and free to move, as soon as it applies an action rule it leaves space to the other modules of its column to activate and move progressively; the first module, followed by the others of its column, moves on the top of the system, North, North-East, East, South-East, South along the rightmost column, and stops reforming a new column. During this movement, the active modules do not interfere one with the other, and move without disconnection nor conflicts, as we have seen in the previous results. As soon as these modules make their way to the right, the second leftmost column becomes the first, its modules are free to change their states and move in order, and the motion is repeated. As there is always a leftmost column, by induction on the number of columns we can conclude that there is always some rule applicable to some module of the system. □

Proposition 17 *The rules actually produce the locomotion of the system from West to East on free ground.*

Proof: As there is always one rule applicable to the system because of Proposition 16, the only thing we need to prove is that the rules do not produce an alternation of opposite movements without moving the system in any direction. The rules that produce a change in the states or in the counters cannot produce any kind of oscillation. The action rules that do not produce a movement in the East direction are North, South, North for bridges and North-East for bridges; we can easily notice from the preconditions of the rules that any module which performs a movement in the North direction cannot produce one in the South

direction until it has reached the opposite side of the column it is attached to; the same can be said for the rest of the rules, so we cannot have an oscillation produced by the application of these rules. As all the other action rules have a component on the East direction, the application of these rules produces an overall movement towards East, without any risk of oscillations. As each time a column is formed again the relative position of the columns and their height is conserved, we can say that the locomotion is an in-shape locomotion. □

We have proved the following:

Theorem 4 *The rules described in this chapter allow any histogram configuration of modules to advance eastwards on a free ground, while keeping the connected shape of the system.*

We have already commented the reconfiguration phase and its correctness in Section 7.6; as the locomotion preserves the shape of the histogram, stopping the system after a given number of rounds and making the left bridges reconfigure into the last columns is enough to produce a reconfiguration of the system into the initial histogram.

7.8 Complexity

Neighbourhood During the locomotion, most of the rules require a module to be able to check the situation of the grid cells of its first neighbor and of some grid cells of its second neighborhood, and to check if these cells are either empty or occupied by another module; in this last case, it has to be able to obtain information about the state of such module. The checking of the second neighborhood modules is needed in order to avoid collisions between active modules and moving bridges: as a bridge moves, its neighborhood needs to be free of active modules willing to move to its goal grid cell; this is achieved by the generation of a delay between two consecutive active modules. No information about any other position is needed.

Memory and computation Locomotion only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to check if preconditions of rules are fulfilled, to memorize fixed values in some counters, and to make simple operations with counters at each step.

Number of moves Let h be the height of the highest column of the histogram, and m the number of its columns. The configuration which requires the maximum number of moves for a module is the one depicted in Figure 7.11; in this configuration, in fact, the number of modules over which the current one needs to walk two or more times is maximal: both sides of each column of height h are free, and the number of columns that reach the maximal height is maximal, given that a separation of at least two cells between them is required in order to avoid the formation of bridges. Moreover, the current module can reach all the rest of the system, so even the modules used as a separation between the maximal columns generate the maximal number of moves. From the moment it activates on the left of the system to its inactivation in the rightmost column, we can see that

each module performs $\approx \frac{m-2}{3}(2h+1)+h$ moves; the number of moves is then $O((m+1)h)$ for each round performed by the system. The number of rounds k (in this case a round is a complete reconfiguration, i.e. we count a round each time the leftmost column is again column number 1) depends linearly on the distance between the starting position and the goal position.

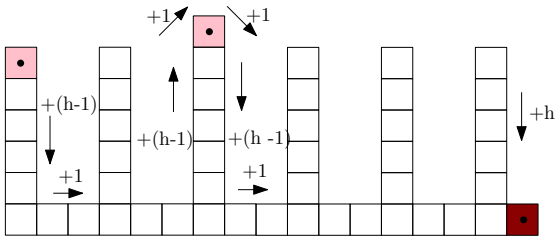


Figure 7.11: Example of a worst case for the initial configuration of an histogram with m column and maximum height h: there are no bottlenecks of width 1 between the columns, so no bridge can be created, and the histogram is formed by columns of height h separated by spaces of width 2. We can observe the steps performed by the bottommost module of the first column, from its activation to its change of state on the right part of the system.

Communication An exchange of information between the modules of the system is performed in order to avoid modifications of the relative position of the columns; a constant size communication is performed each time an exchange rule is applied. During a complete round of the system a module is able to perform at most $O(n)$ exchanges of communication. In fact, the situation in which a module performs the maximal number of communication exchanges during a round of the system is the one in which a module sets its state to *bridge*, exchanges its counters with $O(n)$ modules while they walk over it, activates and overpass again the same amount of modules performing another $O(n)$ exchanges; we can easily notice that, while this happens, the system performs a complete round: in fact, the first $O(n)$ exchanges take place as all the columns on the left of the bridge activate and walk over the bridge, and the second $O(n)$ are performed when the bridge, and the column it connects, activate and overpass again all the rest of columns. If k is the number of complete rounds of the system, $O(kn)$ is the maximum number of communication exchanges that can be performed by a module during the locomotion of the system; the number of rounds k depends linearly on the number of grid cells separating the starting and the goal position.

Number of time steps The computation of the number of time steps can be done as in the case of the free locomotion of the rectangle, by charging to each column the time steps starting from the inactivation of its bottommost module to the inactivation of the bottommost module of the following column. Let n_i be the number of modules of column i, and n the number of modules of the system. The modules of column i start their locomotion by changing their state from *stop* to *inactive*, one after the other,

in n_i time steps. As the bottommost module is inactive, it activates and moves, followed by the others; each module waits to activate until its first and second neighbor are free from other active modules, so a bounded delay between the activation of one module and the activation of the following one is generated; given the configuration of the system, and the fact that active modules can only move North, East and South following the shape of the columns, the time steps between two consecutive activations is always smaller than the total number of grid cells forming the first and the second neighborhood (active modules cannot stay still in a cell, nor move backwards and repeat the same movements, so they cannot occupy the same grid cell two times while walking); as this is a constant fixed number, we charge $O(n_i)$ time steps to column i for its activation; as the total number of time steps for the movement of each column is then $O(n_i)$, the system performs a complete round in $O(n)$ time steps. As before, the number of rounds performed depends linearly on the distance covered by the system during the locomotion.

7.9 Rules

The algorithm is based on 25 different rules; before starting the locomotion each module has the following information stored in its counters: the number of its the column in the order defined (C01), the height of its column (C02), the number of the columns of the histogram (C13), the height of the last column (C15) and the number of rounds (C10).

Rule Name: North
Priority: 6
Preconditions: !SSTOPP !SBRIDG !SSTILL N0*1* T1,1,STOPP !T0,2,ACTIV !T1,2,ACTIV
Postconditions: P0,1 SACTIV A1*1* C007 + 0000 0000 C017 + 0000 0000

- **Rule**: North
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL: it is not applicable to modules set in the *stop*, *bridge* or *still* state.
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
T1,1,STOPP: the module in position $(1, 1)$ needs to be set on *stop*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches to its new north and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets its C07 counter to zero.
C017 + 0000 0000: the module sets its C07 counter to zero.

Rule Name: North-east
Priority: 6
Preconditions: !SSTOPP !SSTILL !SBRIDG N0*1* E1,1 T1,0,STOPP !T2,1,RINFO !T2,0,RINFO !T2,1,ACTIV !T1,2,ACTIV !T2,2,ACTIV E1,2
Postconditions: SACTIV P1,1 A***1 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: North-east
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL: it is not applicable to modules set in the *stop*, *bridge* or *still* state.
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
E1,1: the grid cell in position $(1, 1)$ needs to be empty.
E1,2: the grid cell in position $(1, 2)$ needs to be empty.
T1,0,STOPP: the module in position $(1, 0)$ needs to be set on *stop*.
!T2,1,RINFO: the module in position $(2, 1)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2, 2)$ cannot be set on *active*.
- **Postconditions**:

P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets its C07 counter to zero .
C017 + 0000 0000: the module sets its C07 counter to zero.

Rule Name: East Priority: 7 Preconditions: !SSTOPP !SSTILL !SBRIDG N0*01 !E1,-1 !T1,-1,MOVES !T1,-1,ACTIV !T1,1,ACTIV !T2,0,ACTIV !T2,1,ACTIV !T2,-1,ACTIV !T2,0,RINFO !T2,-1,RINFO !T1,-1,RINFO !(SINACT T1,-1,BRIDG) Postconditions: SACTIV P1,0 A**11 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: East
- **Priority**: 7
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL: it is not applicable to modules set in the *stop*, *bridge* or *still* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north nor a east neighbor.
!E1,-1: the cell grid in position $(1, -1)$ needs to be occupied.
!T1,-1,MOVES: the module in position $(1, -1)$ cannot be set on *moves*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(2, -1)$ cannot be set on *active*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,-1,RINFO: the module in position $(2, -1)$ cannot be set on *rinfo*.
!T1,-1,RINFO: the module in position $(1, -1)$ cannot be set on *rinfo*.
!(SINACT T1,-1,BRIDG): the rule is not applicable by an inactive module if the position $(1, -1)$ is occupied by a bridge.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets its C07 counter to zero.
C017 + 0000 0000: the module sets its C07 counter to zero.

Rule Name: South Priority: 6 Preconditions: !SSTOPP !SBRIDG !SSTILL N*100 !E-1,-1 !T0,-2,ACTIV !T0,-2,MOVES !T1,-1,ACTIV !(T1,-2,ACTIV T0,-2,STOPP) !T0,-2,RINFO !T1,-2,RINFO !(T0,-2,BRIDG T1,-1,STOPP) Postconditions: P0,-1 SACTIV A*111 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: South
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL: it is not applicable to modules set in the *stop*, *bridge* or *still* state.
N*100: the rule concerns only the modules with a west neighbor, and without a east nor a south neighbor.

!E-1,-1: the cell grid in position $(-1,-1)$ needs to be occupied.
!T0,-2,ACTIV: the module in position $(0,-2)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1,-1)$ cannot be set on *active*.
!T0,-2,MOVES: the module in position $(1,-1)$ cannot be set on *moves*.
!T0,-2,RINFO: the module in position $(0,-2)$ cannot be set on *rinfo*.
!T1,-2,RINFO: the module in position $(2,-1)$ cannot be set on *rinfo*.
!(T1,-2,ACTIV T0,-2,STOPP): if the module in position $(1,-2)$ is set on *active* and the module in position $(0,-2)$ is set on *stop* the rule cannot be performed.
!(T0,-2,BRIDG T1,-1,STOPP): if the module in position $(0,-2)$ is set on *bridge* and the module in position $(1,-1)$ is set on *stop* the rule cannot be performed
- Postconditions:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A*111: the module attaches to its new south and east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C007 + 0000 0000: the module sets its C07 counter to zero.
C017 + 0000 0000: the module sets its C07 counter to zero.

Rule Name: South-east Priority: 6 Preconditions: !SSTOPP !SBRIDG !SSTILL N0*01 E1,-1 !T0,-1,BRIDG !T0,-1,ACTIV !T1,-2,ACTIV !T2,0,ACTIV !T1,-2,RINFO !T2,0,RINFO !(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP) !T2,-1,ACTIV !T1,-2,MOVES Postconditions: P1,-1 SACTIV A*111 C007 + 0000 0000 C017 + 0000 0000

- Rule: South-east
- Priority: 6
- Preconditions:
!SSTOPP !SBRIDG !SSTILL: it is not applicable to modules set in the *stop*, *bridge* or *still* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north nor a east neighbor.
E1,-1: the cell grid in position $(1,-1)$ needs to be empty.
!T0,-1,BRIDG: the module in position $(0,-1)$ cannot be set on *bridge*.
!T0,-1,ACTIV: the module in position $(0,-1)$ cannot be set on *active*.
!T1,-2,ACTIV: the module in position $(1,-2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(2,-1)$ cannot be set on *active*.
!T1,-2,RINFO: the module in position $(1,-2)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(2,0)$ cannot be set on *rinfo*.
!T1,-2,MOVES: the module in position $(1,-2)$ cannot be set on *moves*.
!(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP): if the module in position $(1,-2)$ is set on *bridge* and the modules in position $(2,-1)$ and $(0,-1)$ are set on *stop* the rule cannot be performed.
- Postconditions:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*111: the module attaches to its new south and east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C007 + 0000 0000: the module sets its C07 counter to zero.
C017 + 0000 0000: the module sets its C07 counter to zero.

Rule Name: North-east activation for bridges
Priority: 15
Preconditions: SBRIDG N011* E1,1 !T-1,1,ACTIV
!T-1,2,ACTIV !T0,2,ACTIV !T1,0,ACTIV T-1,-1,INACT
Postconditions: P1,1 SACTIV A**11

- **Rule**: North-east activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
E1,1: the cell grid in position $(1, 1)$ needs to be empty.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbors; if it was attached before and if still possible, it attaches to the other neighbors.

Rule Name: North activation for bridges
Priority: 15
Preconditions: SBRIDG N011* !E1,1 !T-1,1,ACTIV !T1,1,ACTIV !T0,2,ACTIV
!T-1,2,ACTIV T-1,-1,INACT
Postconditions: P0,1 SACTIV A**1*

- **Rule**: North activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
!E1,1: the grid cell in position $(1, 1)$ needs to be occupied.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A**1*: the module attaches to its new east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule Name: North for bridges
Priority: 30
Preconditions: SBRIDG N011* T1,1,STOPP T-1,1,STOPP
Postconditions: P0,1 A**1*

- **Rule**: North for bridges
- **Priority**: 30
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,1,STOPP: the module in position $(1, 1)$ needs to be set on *stop*.
T-1,1,STOPP: the module in position $(-1, 1)$ needs to be set on *stop*.
- **Postconditions**:
P0,1: the module moves north.
A**1*: the module attaches to its new east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.

Rule Name: Bridge Formation
Priority: 10
Preconditions: SACTIV N011* T1,0,STOPP T-1,0,STOPP
Postconditions: SBRIDG

- **Rule**: Bridge formation
- **Priority**: 10
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,0,STOPP: the module in position $(1, 0)$ needs to be set on *stop*.
T-1,0,STOPP: the module in position $(-1, 0)$ needs to be set on *stop*.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.

Rule Name: Stop bottommost module
Priority: 9
Preconditions: SACTIV N0100 !(!T-1,0,INACT !T-1,0,STOPP) E-1,-1
Postconditions: SSTOPP C006 + 0000 0001

- **Rule**: Stop bottommost module
- **Priority**: 9
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N0100: the rule concerns only the modules with a west, and without other neighbors.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty. !(!T-1,0,INACT !T-1,0,STOPP): the cell grid in position $(-1, 0)$ needs to be occupied by a module, either set in the *inactive*either in the *stop* state.
- **Postconditions**:
P0,1: the module moves north.
SSTOPP: the module changes its state to *stop*.
A**1*: the module attaches to its new east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C006 + 0000 0001: the module sets its C06 counter to 1.

Rule Name: Stop other modules
Priority: 9
Preconditions: SACTIV N0*01 T0,-1,STOPP !V0,-1,C001 C001 W0,-1,C001 C001 V0,-1,C006 C002
Postconditions: SSTOPP C006 + 0,-1,C006 0001

- **Rule**: Stop other modules
- **Priority**: 9
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north nor west neighbor.
(!V0,-1,C001 C001 W0,-1,C001 C001): the value of the counter C01 of the module has to coincide with the counter *C*01 of its south neighbor.
V0,-1,C006 C002: the value of the counter C02 of the module has to be bigger than the value of the counter C06 of its south neighbor.
- **Postconditions**:
P0,1: the module moves north.
SSTOPP: the module changes its state to *stop*.
C006 + 0,-1,C006 0001: the module checks the value of the counter C06 of its south neighbor, increases it by 1 and sets its own C06 counter to this result.

Rule Name: Locomotion break
Priority: 100
Preconditions: SSTOPP N*0*0 = C001 0001 = C004 C010
Postconditions: SSTILL

- **Rule**: Locomotion break
- **Priority**: 100
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N*0*0: the rule concerns only the modules without a east or south neighbor.
= C001 0001: the rule applies only if the value of the counter C01 of the module equals 1.
= C004 C010: the rule applies only if the value of the counter C04 of the module equals the value of C10.
- **Postconditions**:
SSTILL: the module changes its state to *still*.

Rule Name: Bottommost module inactivation
Priority: 10
Preconditions: SSTOPP N*0*0 E-1,-1 E-1,1 !T0,1,ACTIV !T1,1,ACTIV !T-1,2,ACTIV !T2,1,ACTIV !T1,2,ACTIV
Postconditions: SINACT C004 + C004 0001

- **Rule**: Bottommost module inactivation
- **Priority**: 10
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N*0*0: the rule concerns only the modules without a west nor a south neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.
C004 + C004 0004: the module adds 1 to its C04 counter.

Rule Name: Other modules inactivation
Priority: 10
Preconditions: SSTOPP N10*1 E-1,-1 T0,-1,INACT E-1,0 E-1,1 !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV
Postconditions: SINACT C004 + C004 0001

- **Rule**: Other modules inactivation
- **Priority**: 10
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N10*1: the rule concerns only the modules with a north and a south neighbor, and without a west neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,0: the grid cell in position $(-1, 0)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
T0,-1,INACT: the module in position $(0, -1)$ needs to be inactive.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2, 2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.
C004 + C004 0001: the module adds 1 to its C04 counter.

Rule Name: Topmost module inactivation
Priority: 12
Preconditions: SSTOPP N00*1 E-1,-1 T0,-1,INACT !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV !T1,2,ACTIV !T2,2,ACTIV
Postconditions: SINACT C004 + C004 0001

- **Rule**: Topmost module inactivation
- **Priority**: 12
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N00*1: the rule concerns only the modules with a south neighbor, and without a north nor a west neighbor.
E-1,-1: the grid cell in position $(-1,-1)$ needs to be empty.
T0,-1,INACT: the module in position $(0,-1)$ needs to be inactive.
!T1,0,ACTIV: the module in position $(1,0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1,1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1,-1)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2,1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(2,1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1,2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2,2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.
C004 + C004 0001: the module adds 1 to its C04 counter.

Rule Name: Exchange rule for Bridges
Ponte su cui arriva un attivo Priority: 100
Preconditions: SBRIDG !W0,1,C001 C001 !E0,1 = C007 0000
Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000 C017 + 0,1,C004 0000 C007 + 0000 0001

- **Rule**: Exchange rule for Bridges
- **Priority**: 100
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0,1)$ needs to be occupied.
!W0,1,C001 C001: the value of the counter C01 of the module has to be less than the value of the counter C01 of its north neighbor.
= C007 0000: the rule applies only if the value of the counter $C07$ of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its north neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its north neighbor into its own counter C12.
C017 + 0,1,C004 0000: the module stores the value of the counter C04 of its north neighbor into its own counter C17.
C007 + 0000 0001: the modules sets its counter C07 to 1.

Rule Name: Exchange rule for Actives
Priority: 100
Preconditions: T0,-1,BRIDG V0,-1,C001 C001 = C007 0000
Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000 C017 + 0,-1,C004 0000 C007 + 0000 0001

- **Rule**: Exchange rule for Actives
- **Priority**: 100
- **Preconditions**:
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.
V0,-1,C001 C001: the value of the counter C01 of the module has to be bigger than the value of the counter C01 of its south neighbor.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C017 + 0,-1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets its counter C07 to 1.

Rule Name: Internal change of counters I
Priority: 100
Preconditions: SRINFO E0,1 = C007 0001
Postconditions: C001 + C011 0000 C002 + C012 0000 C004 + C017 0000 SACTIV

- **Rule**: Internal change of counters I
- **Priority**: 100
- **Preconditions**:
SRINFO: it is applicable only to modules set in the *rinfo* state. E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.
C004 + C017 0000: the module stores the value of the counter C17 of its south neighbor into its own counter C04.

Rule Name: Internal change of counters II
Priority: 100
Preconditions: SRINFO !E0,1 = C007 0001
Postconditions: C001 + C011 0000 C002 + C012 0000 C004 + C017 0000 SBRIDG

- **Rule**: Internal change of counters II
- **Priority**: 100
- **Preconditions**:

SRINFO: it is applicable only to modules set in the *rinfo* state. !E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.
C004 + C017 0000: the module stores the value of the counter C17 of its south neighbor into its own counter C04.

Rule Name: Case of the last column (Bridge) Priority: 100 Preconditions: SBRIDG T0,1,ACTIV = C001 C013 = C007 0000 !(!V0,1,C001 C013 W0,1,C001 C013) Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000 C017+ 0,1,C004 0000 C007 + 0000 0001

- **Rule**: Case of the last column (Bridge)
- **Priority**: 100
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
T0,1,ACTIV: the module in position $(0, 1)$ needs to be set on *active*.
!(!V0,1,C001 C013 W0,1,C001 C013): the value of the counter C01 of the module has to be different from the value of the counter C13 of its north neighbor.
= C001 C013: the rule applies only if the value of the counter C01 of the module is equal to the value of its counter C13.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C017 + 0,1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Case of the last column (Actives) Priority: 100 Preconditions: SACTIV T0,-1,BRIDG ¡ C001 C013 !V0,-1,C001 C013 W0,-1,C001 C013 = C007 0000 Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000 C017 + 0,-1,C004 0000 C007 + 0000 0001

- **Rule**: Case of the last column (Actives)
- **Priority**: 100
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.

(!V0,-1,C001 C013 W0,-1,C001 C013): the value of the counter C01 of the module has to be equal to the value of the counter C13 of its south neighbor.
< C001 C013: the rule applies only if the value of the counter C01 of the module is less than the value of its counter C13.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- Postconditions:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C017 + 0,-1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Counter C007 Priority: 15 Preconditions: SBRIDG E0,1 !(=C017 0000 =C007 0000) Postconditions: C007 + 0000 0000 C017 + 0000 0000

- Rule: Counter C007
- Priority: 15
- Preconditions:
SBRIDG: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
< C001 C013: the rule applies only if the value of the counter C01 of the module is less than the value of its counter C13.
!(=C017 0000 =C007 0000): the rule does not apply to modules with the two counters C17 and C07 set to zero.
- Postconditions:
C017 + 0000 0000: the modules sets its counter C17 to 0.
C007 + 0000 0000: the modules sets its counter C07 to 0.

Rule Name: Counting of the modules of the last column Priority: 100 Preconditions: SBRIDG !E0,1 = C013 C001 = C016 0000 !V0,1,C004 C010 W0,1,C004 C010 Postconditions: C014 + C014 0001 C016 + 0000 0001

- Rule: Counting of the modules of the last column
- Priority: 100
- Preconditions:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
= C013 C001: the rule applies only if the value of the counter C01 of the module equals the value of its counter C13.
= C016 0000: the rule applies only if the value of the counter C16 of the module is 0.
(!V0,1,C004 C010 W0,1,C004 C010): the value of the counter C10 of the module has to be equal to the value of the counter C04 of its north neighbor.
- Postconditions:
C014 + C014 0001: the module adds 1 to its C04 counter.
C016 + 0000 0001: the module sets its C16 counter to 1.

Rule Name: Auxiliar counter for the counting
Priority: 100
Preconditions: SBRIDG E0,1 = C016 0001
Postconditions: C016 + 0000 0000

- **Rule**: Auxiliary counter for the counting
- **Priority**: 100
- **Preconditions**:
SBRIDG: the module changes its state to *bridge*.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C016 0001: the rule applies only if the value of the counter C16 of the module is 1.
- **Postconditions**:
C016 + 0000 0000: the module adds 0 to its C04 counter.

Rule Name: Bridges reconfiguration
Priority: 100
Preconditions: SBRIDG E0,1 = C014 C015
Postconditions: SMOVES

- **Rule**: Bridges reconfiguration
- **Priority**: 100
- **Preconditions**:
SBRIDG: the module changes its state to *bridge*.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C014 C015: the rule applies only if the value of the counter C14 of the module equals the value of its counter C15.
- **Postconditions**:
SMOVES: the module changes its state to *moves*.

Chapter 8

Histogram locomotion with inferior obstacles

8.1 Goal

The purpose of the set of rules presented in this chapter is to produce the eastward locomotion of any modular robotic system initially configured as a connected histogram, in the presence of inferior obstacles, i.e. obstacles that lay on the ground. In our settings, obstacles lay on the ground (a horizontal line) and are configured as histograms.

8.2 Strategy

The strategy is similar to the one of the free locomotion, but with some adjustments that make it possible to crawl over the obstacles. Before reaching an obstacle, the histogram moves under the original locomotion rules described in Chapter 7; as it encounters the obstacle, some module of the system create a static path over it, allowing the others to walk over and cross it without any disconnection. This path is created through the introduction of a new state for the modules: the *path* state. We can observe the process of the formation of a path depicted in Figure 8.1.

As the first active module of the system reaches the ground after crossing the obstacle, it changes its state to *stop*, and starts the reconfiguration of its column. During the locomotion of the active modules over the paths no exchange of information is performed, so the first column that is reconfigured after the obstacle is in general not the one that follows in the order of the columns; moreover, it is possible that some modules of such column are part of the path, and that the column reconfigures only partially, until the path reactivates. As soon as all the modules have crossed the obstacle, one by one the modules of the path change their state to active again, and walk until they reach the rest of the system and reconfigure into their columns. After the crossing, the system recreates its initial shape, and the locomotion can continue as in the case of the free locomotion. The strategy present some analogies with the strategy for the rectangle locomotion over high obstacles: in the case of the rectangle, when the

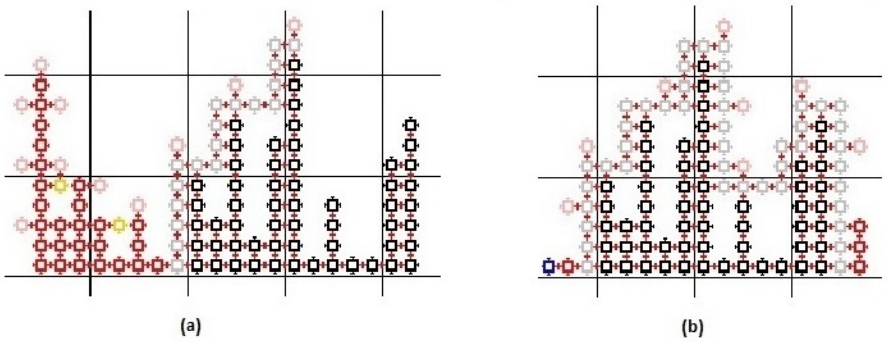


Figure 8.1: Example of the formation of a path: as the modules encounter an obstacle they change their state to *path*, and stay still until the rest of the system has crossed the obstacle. Grey modules are *path* modules, active modules are pink, red modules are in the *stop* state and yellow modules are bridges. Obstacles are depicted in black.

system encounters an high part of an obstacle it turns into a connected worm, allowing the active modules to pass by; such connected worm is transformed into a connected path in the case of the histogram: the need of the introduction of a new state is the need to avoid the comparison between the counters of the active modules and the rest of modules during the overpassing of the obstacle. The shape of the histogram, unlike in the case of the rectangle, is not conserved neither in the high nor in the low parts of the obstacles; this is a choice done in order to maintain the rules as simple as possible, as conserving the shape of the histogram in the low parts would produce a significant complication of the rules, but no significant advantages.

8.3 Locomotion rules

The locomotion of the system before the crossing of the obstacle is mostly produced through the rules used in the free locomotion of the histogram, adjusted to the introduction of the new states *obstacle* and *path*. We can see an example of such adjustments in Figure 8.2, which depicts the new action rule North, and compare it with the former North rule (refer to Chapter 7). For the details of the modifications of the remaining rules, see the detailed rules in Section 8.9.

As the obstacle is encountered, some additional rules are needed in order to avoid disconnections. During the locomotion, columns start the movement from their topmost module, and the other modules progressively activate until the bottommost module moves and gives space to the following column to start the inactivation; this process is now reversed for the column immediately on the left of the obstacle, as we can observe in Figure 8.3.

Such inversion is achieved through the introduction of the two inversion rules depicted in Figure 8.4; as the bottommost module of the column changes its state to *inactive*, it activates applying the North-West inversion rule, walks on the left side of its column through the North rule, and then crosses it through

North Rule
Priority: 6
Preconditions:
State≠(Stop,Still,Bridge,Obsta,Paths) & (0,1)=Empty & (1,0)=Full & State(0,2)≠Active & State(1,2)≠Active & State(1,1)=(Stop,Paths,Inact) & Not(State=Inactive,State(-1,0)=Active,(-2,0)=Empty) & Not(State=Inactive,State(1,0)=Paths,((-2,0) or (0,-1))= Full)
Postconditions:
Move (0,1) & State=Active& C007 = 0000 & C017 = 0000

Figure 8.2: North rule for the histogram locomotion with obstacles: we can notice some modification with respect to the North rule of the free locomotion; new preconditions as *State≠Obsta* are introduced in order to deal with the new states, while others as *Not(State=Inactive, State*$(-1,0)$*=Active,*$(-2,0)$*=Empty)* deal with the new configurations that are generated by the crossing of the obstacle

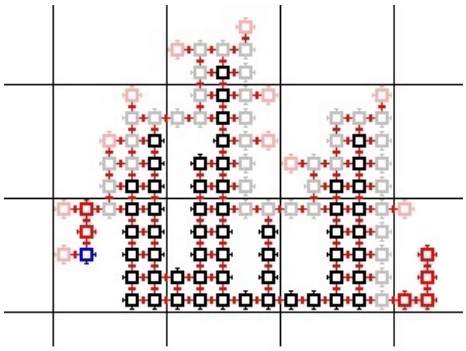


Figure 8.3: Histogram locomotion with obstacles: the last column on the left of the obstacle inverts its movement in order to avoid disconnections. The first module to move is now the bottommost one.

the North-East inversion rule.

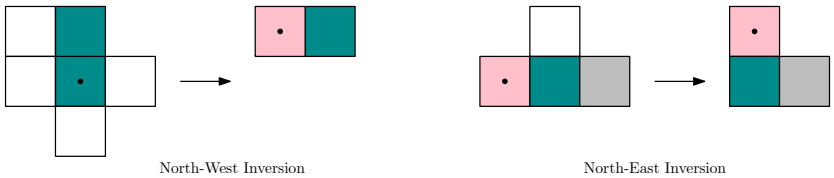


Figure 8.4: Inversion rules for the last column.

Notice that this type of inversion cannot be applied to the case of an histogram formed by a unique column, as the North-West Inversion would interfere with the locomotion; this case needs some special modification that will be treated in Section 8.6.

8.4 Path formation and reactivation

The formation of the path over the obstacles is achieved through the application of the six rules depicted in Figures 8.5 and 8.6. As a module encounters an obstacle, it changes its state to *paths* through the application of one of the rules depicted in Figure 8.5, and conserves its position until all the active modules have crossed the obstacle.

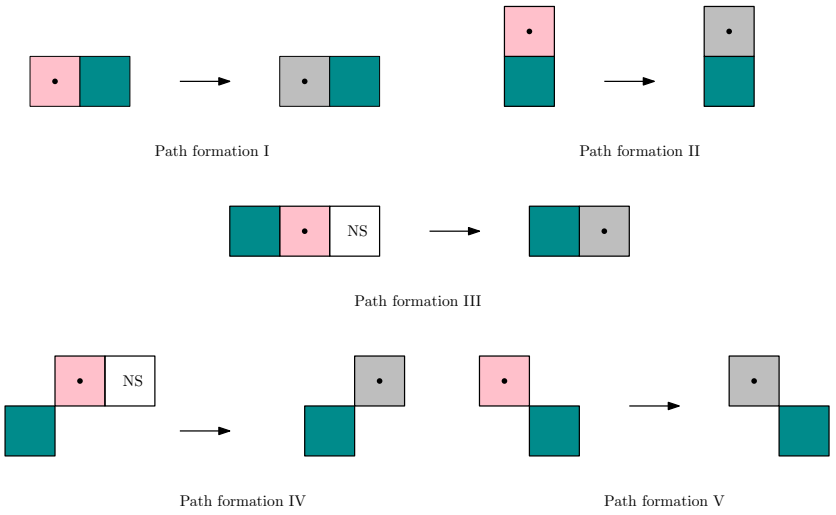


Figure 8.5: Rules for the formation of the path: the current module is indicated with a dot. Colored grid cells are occupied, pink cells are modules set in the *active* state, gray cells are modules of the path, and blue cells are obstacles. NS indicates that if a module occupies the grid cell, it cannot be set to *stop*.

As none of the action rules created for the free locomotion produces a movement with a component in the west direction, an additional rule is needed in order to reach all the grid cells of the neighborhood of the obstacle and form a complete and connected path. The rule depicted in Figure 8.6 solves this problem; this rule belongs to the path rules set, but differs from the others as it produces a South-West movement, apart from the change of state.

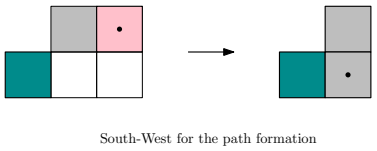


Figure 8.6: South-West rule for the path formation.

The reactivation of the path is done through the six activation rules depicted in Figure 8.7. Notice that the graphic representation does not contain all the preconditions of the rules, as some of them are too complex to be expressed in this way. The full description of the rules can be found in Section 8.9.

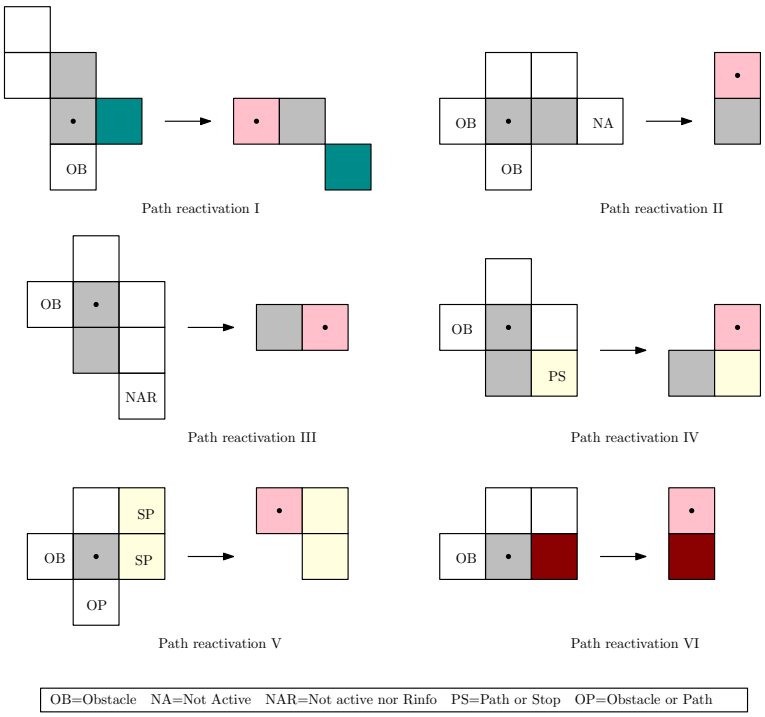


Figure 8.7: Path reactivation rules: as always, the current module is indicated with a dot. Colored grid cells are occupied, pink cells are modules set in the *active* state, gray cells are modules of the path, blue cells are obstacles and red cells are modules set on the *stop* state.

8.5 Reconfiguration

As the system always conserves the order and the height of the columns during the locomotion, the idea of the reconfiguration is to stop the locomotion when column 1 is again the leftmost column of the system, as in the case of the free locomotion. In the new settings, however, counting the number of inactivations of the modules is not a suitable solution, as modules do not inactivate the same number of times during the locomotion: the path stays still while the rest continue activating and inactivating, so a difference between the value of the counter is produced and the former system for the final reconfiguration does not work anymore. The counters used for the new reconfiguration process are C09, C10 and C18. C09 is a measure of the position of the modules on the x-axis; initially set as equal to the column counter (1 for the first column, 2 for the second one, and so on), it is increased (decreased) by one each time a movement with an east (west) component is performed; the position of each module is then controlled by its horizontal distance from the initial position of the first column.

C10, fixed before the locomotion starts, stores the x-axis position in which the system will start its reconfiguration. As soon as the x-position stored in C10 is reached, and the leftmost column is again column 1, its bottommost module

sets its state to *Still*, blocking the movement of the first column.

As the modules of the last column walk on the blocked column 1, they change their C18 counter to 1 and start the counting phase for the reactivation of the bridges, as in the free locomotion case. The rules for the counting phase and the reactivation of bridges can be seen in detail in Section 8.9.

8.6 Case of one column

The one column version of these rules constitutes an alternative set of rules for the locomotion of the rectangle with one column; in the presence of many obstacles, such rules are preferable to the set of rules presented in Chapter 2, as they avoid the reconfiguration of the system over the obstacle, making the crossing fluider and faster.

In this particular case, in order to maintain the stability of the system and avoid disconnections, the process of the formation of the path is modified: the construction starts now when the first module reaches the bottommost position on the left of the obstacle. In this way the leftmost side of the obstacle is completely recovered by the path, as we can observe in Figure 8.8, and modules are free to walk over it avoiding the last column inversion, impossible for the case of one column.

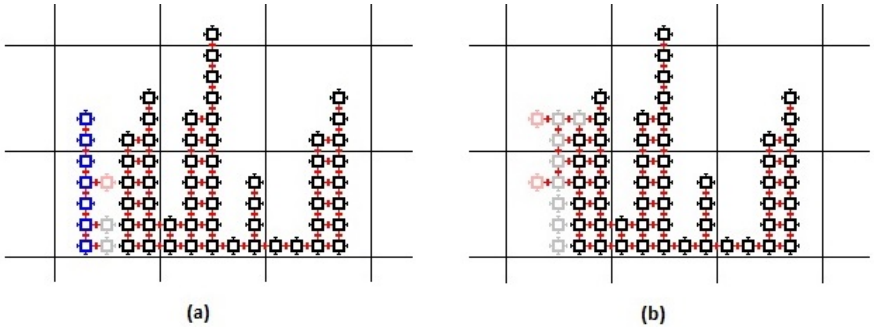


Figure 8.8: Histogram locomotion with obstacles in the case of one column: there is no inversion for the last column, as the path covers the obstacle completely.

Apart from a few adjustments of some preconditions and priorities, the modification of the rules mostly consists in the elimination of the *North-West Inversion* rule, and in the introduction of two new rules: the "South rule for path modules" and the "First path module" formation. All the rules for the case of one column are detailed in Section 8.9.

8.7 Correctness

The modifications applied to the existent free locomotion rules are operated in order to adapt the same rules to the new states of the modules and to some new particular cases; all the results proved in the previous chapter are still valid for

the new version of the rules, and we will not prove it again here. In this section we will use such results and examine the impact of the new defined rules on the correctness conclusions.

As always, we will prove that:

1. The system stays connected during the whole movement.
2. The new rules do not produce collisions during the movement.
3. There is always at least one rule that is applicable to some module of the system.
4. The rules produce a locomotion of the system from West to East in the presence of obstacles.

Proposition 18 *The system stays connected during the whole movement, i.e. no rule produces a disconnection between the module applying it and the system.*

Proof: In Chapter 7 we have proved that the system stays connected during the free locomotion. As the obstacle is encountered, a connected path is created over the obstacle, and active modules walk over it until they reach the right part of the obstacle and reconfigure again. As soon as the path modules are free to move again, they activate one by one and walk over the remaining path, until the last path module is active and the obstacle is crossed, therefore connection is guaranteed by the path structure. □

Lemma 6 *Each module can apply at most one action rule at a time.*

Proof: We have already proved in the previous chapter that the set of rules that produce the free locomotion of the system satisfies this property; the only thing to check now is that none of the new action rules interferes with the preexistent rules, or with another new rule.

The new action rules introduced are the *South-West rule for the path formation*, the *North-West inversion* and the *North-East inversion rules*; the priority of these three rules are different from all the other action rules, so there is no possible interaction between them; moreover, the priority of the South-West rule is different from the priority of the inversion rules, so the only possible interference can be generated between the two inversion rules; we can easily exclude this possibility too.

As the North-East action rule requires the relative position $(0, 1)$ to be empty, while the North-West inversion requires the same position to be occupied, the rules are incompatible, and each module can apply at most one action rule at a time. □

Proposition 19 *No collisions are produced during the movement.*

Proof: We want to prove that two different modules never move to the same grid cell at the same time while applying the rules. Due to Lemma 6 we only need to worry about conflicts created by the application of one action rule at a time. As we have proved that the free locomotion rules do not generate any collisions, due to their specific preconditions, we want now to exclude the case of a collision generated during the crossing of the obstacle, or between free

locomotion modules and obstacle traversing modules. We analyze the North-west inversion rule, under the assumption that the system is crossing an obstacle for the first time; as the free locomotion does not generate any disconnection and is performed conserving the shape of the system, we can reasonably suppose that the system is connected and still in the form of an histogram in its left part.

While the current module is applying the North-west inversion rule, the only possibility is that its column is the leftmost of the system; the only module that could cause a collision would be either an active module attached to the column, or and inactive or stopped module which belongs to the column. The only possibility for the first case is a module in position A applying the South rule (refer to Figure 8.9); we can easily see that this is impossible, as a precondition for the application of the South rule is that the relative position $(1, 0)$ is empty; this would mean that the module is connected to another column on its left side, which is a contradiction.

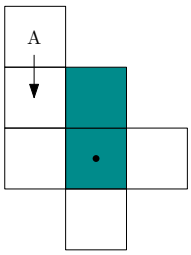


Figure 8.9: The current module applies the South-west rule, while a module in position A applies the South rule generating a collision.

The second possibility is impossible too, as the only rules that imply a movement in the west direction are the North-west rule, which has a north component too and does not suit our case, and the South-west rule for the formation of the path, which is applicable only to active modules.

Such contradictions can be found in all the new rules, so no collisions are produced. As Proposition 18 states that the system is always connected, the crossing of the first obstacle does not generate any disconnection, and the conclusions are true for any obstacle encountered. □

Proposition 20 *There is always at least one rule that is applicable to some module of the system.*

Proof: Before the crossing of the obstacle, the locomotion is performed conserving the shape of the histogram, so there is always a leftmost column of the system; among the modules of the leftmost column there is always a module able to apply a rule: either its topmost module, during the free locomotion, either its bottommost module in the case of the last column inversion. As the obstacle is encountered, as a leftmost column is not present anymore, there is still always a module able to apply a rule: either an active module able to change its state to path, either an active module advancing over the path, or the same modules of the path, which activate again as soon as all the rest of the system has passed by. As the obstacle is crossed, a first column is again formed and the motion is repeated. □

Proposition 21 *The rules actually produce a locomotion of the system from west to east, on free ground and over histogram obstacles.*

Proof: As there is always one rule applicable to the system because of Propopsition 20, the only thing we need to prove is that the rules do not produce an alternation of opposite movements without moving the system in any direction. We have already proved that the free locomotion is correctly performed in this sense, and we prove now that there is no oscillation produced by the new rules introduced. We analyze here the formation of the path; all the other new rules present the same behavior and can be analyzed in an analogous way.

The path formation rules that do not produce any change of position cannot cause any oscillation of the system. The only path rule that causes a movement is the *South-west rule for the formation of the path*. As soon as a module applies such rule it moves South-west, changes its own state to *path*, and stands still until its reactivation is possible. No advance rule can be applied to a path module, so we only need to worry about reactivation rules with an east component in the movement, as they are the only one applicable and that could cause oscillations.

After the application of the South-west rule, the module has a unique neighbor: a north neighbor set on the *path* state. Each one of the activation rules with an east direction is applicable only by modules without a north neighbor; a module that has applied the South west rule, then, has to wait for the rest of the system to move and overpass before applying any other rule; as there is no disconnection during the application of the rules because of Proposition 18, the only possibility for the module to lose its north neighbor is that the system has walked eastward, overpassing it and reaching its left side, so an application of an east rule could not generate any oscillation.

As all the other rules can be treated in the same way, no oscillation can be produced by the application of the rules. □

8.8 Complexity

Neighbourhood As in the case of the free locomotion, the rules require a module to be able to check the situation of the grid cells of its first and second neighbor, and to know if these cells are either empty or occupied by another module; in this last case, to obtain information about the state of such module. The need of information about the second neighborhood is the need to avoid collisions between the moving bridges and the active modules, as in the case of free locomotion. No information about any other position is needed.

Memory and computation As the free locomotion, the locomotion with obstacles only requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to check if preconditions of rules are fulfilled, to memorize a fixed value in some counters and to make simple operations with counters at each step.

Number of moves As we have already analyzed the case of the free locomotion of the histogram, we can now focus on the crossing of the obstacle. Let B be the number of boundary cells of the obstacle, i.e. the set of the grid cells which share at least one edge or a vertex with the obstacle

with a positive abscissa (if we consider the straight line representing the ground as the axis $x = 0$). Each of the modules of the system crosses the obstacle performing at most $O(B)$ moves: in case it takes part in the construction of the path at position k, as in the Figure 8.10, the module performs k moves until it gets to the right position k, changes its state to *path*, stays still until it is time to activate again, and then performs $B - k$ moves walking over the remaining part of the paths; in case the path is already formed when it crosses the obstacle, the module walks over it as an active module from position 1 to position B. Notice that this is an upper bound, as bottlenecks of width 1 and 2 formed by the columns of the obstacle are not filled by the module of the path. The total number of moves for a module then is the sum of the number of moves performed before and after the obstacle, calculated as in the free locomotion case, and at most $O(B_i)$ for each encountered obstacle, where B_i is the number of grid boundary cells of obstacle i.

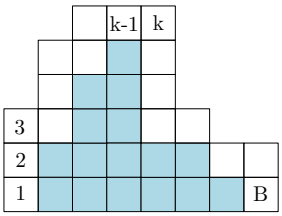


Figure 8.10: Boundary cells of the obstacle: the cells of the boundary are enumerated from left to right. The module in position k is the k-*th* module of the path according to this order.

Communication As no exchange of communication is performed during the crossing of the obstacle, the communication operated by a module is at most $O(n)$ as in the case of the free locomotion, where n is the total number of modules of the system.

Number of time steps As we have already analyzed the number of time steps needed for the free locomotion of the histogram, we focus here on the overpassing of the obstacles. Let M be the number of grid cells of the positive boundary of the obstacle (i.e. the boundary grid cells contained in the positive half-plane $y > 0$, where y-axis is perpendicular to the line representing the ground). If the number n of modules of the system is lower than M, as soon as the modules of the system touch the obstacle they form the path over the first n boundary cells; as they activate again, they walk over and rechange their state to path another time, covering the following n cells on the boundary; this process is repeated until they reach the ground another time. The formation of the path and its reactivation both require $O(n)$ time steps, as at most a constant delay is produced between the activation of two following modules of the system. The number of times this cycle is repeated depends linearly on the number of boundary cells of the obstacle, i.e. on the total number of modules of the obstacle. If $n = M$, $O(n)$ time steps are needed to create the path and $O(n)$ to activate again. If the number of modules is higher than M, $O(M)$ time

steps are needed to create the path over the obstacle, $O(M)$ are needed for its reactivation and an additional $O(n - M)$ is required: before starting the reactivation of the modules, in fact, the path needs to wait that all the active modules that don't take part to the formation of the path have overpassed the obstacle.

8.9 Rules

The algorithm is based on 40 different rules; before starting the locomotion each module has the following information stored in its counters: the number of its the column in the order defined (C01), the height of its column (C02), the number of the columns of the histogram (C13), the height of the last column (C15) and the number of rounds (C10).

Rule Name: North
Priority: 6
Preconditions: !SSTOPP !SBRIDG !SOBSTA !SSTILL !SPATHS N0*1*
!(!T1,1,STOPP !T1,1,PATHS !T1,1,INACT) !T0,2,ACTIV !T1,2,ACTIV !(SINACT T-1,0,ACTIV E-2,0)
!(SINACT T1,0,PATHS !(E-1,0 E0,-1))
Postconditions: P0,1 SACTIV A1*1* C007 + 0000 0000 C017 + 0000 0000

- **Rule**: North
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SPATHS !SOBSTA: it is not applicable to modules set in the *stop*, *bridge*, *still*, *path* or *obsta* state
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
!(!T1,1,STOPP !T1,1,PATHS !T1,1,INACT): the module in position $(1, 1)$ needs to be set on *stop*, *path*, or *inactive*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
!(SINACT T-1,0,ACTIV E-2,0): the rule cannot be applied if the module is set on *inactive*, the module $(-1, 0)$ is active and the position $(-2, 0)$ is not empty.
!(SINACT T1,0,PATHS !(E-1,0 E0,-1)): the rule cannot be applied if the module is set on *inactive*, the module $(-1, 0)$ is active and one of the position $(-1, 0)$ and $(0, -1)$ is not empty.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches to its new north and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

Rule Name: North-east
Priority: 6
Preconditions: !SSTOPP !SOBSTA !SSTILL !SBRIDG !SPATHS N0*1* E1,1 !T1,2,RINFO !(!T1,0,STOPP !T1,0,PATHS) !T2,1,RINFO !T2,0,RINFO !T2,1,ACTIV !T2,2,ACTIV !T1,2,ACTIV !T-1,0,RINFO !T0,2,ACTIV !T1,2,ACTIV !(SINACT T-1,0,ACTIV E-2,0) !(SINACT T1,0,PATHS !(E-1,0 E0,-1))
Postconditions: SACTIV P1,1 A**11 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001

- **Rule**: North-east
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SOBSTA !SSTILL !SBRIDG !SPATHS: it is not applicable to modules set in the *stop*, *obsta*, *still*, *bridge* nor *path* state.
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
E1,1: the grid cell in position $(1, 1)$ needs to be empty.
!T1,2,RINFO: the module in position $(1, 2)$ cannot be set on *rinfo*.
!T-1,0,RINFO: the module in position $(-1, 0)$ cannot be set on *rinfo*.
!(!T1,0,STOPP !T1,0,PATHS): the module in position $(1, 0)$ needs to be set either on *stop* or in the *path* state.
!T2,1,RINFO: the module in position $(2, 1)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2, 2)$ cannot be set on *active*.
!(SINACT T1,0,PATHS !(E-1,0 E0,-1)): the rule cannot be applied if the module is set on *inactive*, the module $(-1, 0)$ is active and one of the position $(-1, 0)$ and $(0, -1)$ is not empty.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: East
Priority: 7
Preconditions: N**01 !SSTOPP !SSTILL !SOBSTA !SBRIDG !SPATHS !E1,-1 !T1,-1,ACTIV !T1,-1,OBSTA !(SINACT T1,-1,BRIDG) !(T0,1,ACTIV E-1,1) !T0,1,INACT !T0,1,STOPP !T1,1,ACTIV !T2,0,ACTIV !T2,1,ACTIV !T2,-1,ACTIV !T2,0,RINFO !T2,-1,RINFO !T1,-1,RINFO
Postconditions: SACTIV P1,0 A**11 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001

- **Rule**: East
- **Priority**: 7
- **Preconditions**:
!SSTOPP !SSTILL !SOBSTA !SBRIDG !SPATHS: it is not applicable to modules set in the *stop*, *obsta*, *still*, *bridge* nor *path* state.
N**01: the rule concerns only the modules with a south neighbor, and without a east neighbor.
!E1,-1: the cell grid in position $(1, -1)$ needs to be occupied.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.

!T1,-1,OBSTA: the module in position $(1, -1)$ cannot be set on *obsta*.
!T0,1,INACT !T0,1,STOPP: the module in position $(0, 1)$ cannot be set on *inactive* nor *stop*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(2, -1)$ cannot be set on *active*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,-1,RINFO: the module in position $(2, -1)$ cannot be set on *rinfo*.
!T1,-1,RINFO: the module in position $(1, -1)$ cannot be set on *rinfo*.
!(SINACT T1,-1,BRIDG): the rule cannot be applied if the module is set on *inactive* and the module $(1, -1)$ is a bridge.
!(T0,1,ACTIV E-1,1): the rule cannot be applied if the module $(0, 1)$ is set on *active* and the grid cell $(-1, 1)$ is empty.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: South Priority: 6 Preconditions: N*100 !SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS !T1,-1,ACTIV !E-1,-1 !T0,-2,ACTIV !(T1,-2,ACTIV T0,-2,STOPP) !T0,-2,RINFO !T1,-2,RINFO !T-1,-1,RINFO !(T0,-2,BRIDG T1,-1,STOPP) !T1,-2,ACTIV Postconditions: SACTIV P0,-1 A***1 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: South
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS: it is not applicable to modules set in the *stop*, *bridge*, *still*, *obsta* nor *path* state.
N*100: the rule concerns only the modules with a west neighbor, and without a east or a south neighbor.
!E-1,-1: the cell grid in position $(-1, -1)$ needs to be occupied.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T0,-2,ACTIV: the module in position $(0, -2)$ cannot be set on *active*.
!T1,-2,ACTIV: the module in position $(1, -2)$ cannot be set on *active*.
!T1,-2,RINFO: the module in position $(1, -2)$ cannot be set on *rinfo*.
!T-1,-1,RINFO: the module in position $(-1, -1)$ cannot be set on *rinfo*.
!T0,-2,RINFO: the module in position $(0, -2)$ cannot be set on *rinfo*.
!(T1,-2,ACTIV T0,-2,STOPP): the rule cannot be applied if the module $(1, -2)$ is set on *active* and the module $(0, -2)$ is set on *stop*.
!(T0,-2,BRIDG T1,-1,STOPP): the rule cannot be applied if the module $(0, -2)$ is set on *bridge* and the module $(1, -1)$ is set on *stop*.
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.

A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

Rule Name: South-east Priority: 6 Preconditions: N0*01 !SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS E1,-1 !T0,-1,BRIDG !T0,-1,ACTIV !T1,-2,ACTIV !T2,0,ACTIV !T1,-2,RINFO !T2,0,RINFO !T2,-1,ACTIV !(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP) Postconditions: P1,-1 SACTIV A*111 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001

- **Rule**: South-east
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS: it is not applicable to modules set in the *stop*, *bridge*, *still*, *obsta* nor *path* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north or a west neighbor.
E1,-1: the cell grid in position $(1, -1)$ needs to be empty.
!T0,-1,BRIDG: the module in position $(0, -1)$ cannot be set on *bridge*.
!T0,-1,ACTIV: the module in position $(0, -1)$ cannot be set on *active*.
!T1,-2,ACTIV: the module in position $(1, -2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T1,-2,RINFO: the module in position $(1, -2)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(1, -2)$ cannot be set on *rinfo*.
!T2,-1,ACTIV: the module in position $(2, -1)$ cannot be set on *active*.
!(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP): the rule cannot be applied if the module $(1, -2)$ is set on *bridge*, the module $(2, -1)$ is set on *stop* and the module $(0, -1)$ is set on *stop*.
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A*111: the module attaches to its new east, west and south neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: Bridge Formation Priority: 10 Preconditions: N011* SACTIV T1,0,STOPP T-1,0,STOPP = C018 0000 Postconditions: SBRIDG

- **Rule**: Bridge formation
- **Priority**: 10
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,0,STOPP: the module in position $(1, 0)$ needs to be set on *stop*.
T-1,0,STOPP: the module in position $(-1, 0)$ needs to be set on *stop*.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.

Rule Name: North-east activation for bridges
Priority: 15
Preconditions: SBRIDG N011* E1,1 !T-1,1,ACTIV !T-1,2,ACTIV !T0,2,ACTIV !T1,0,ACTIV T-1,-1,INACT
Postconditions: P1,1 SACTIV A***1 C009 + C009 0001

- **Rule**: North-east activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
E1,1: the cell grid in position $(1, 1)$ needs to be empty.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: North activation for bridges
Priority: 15
Preconditions: SBRIDG N0*1* !E1,1 !T-1,1,ACTIV !T1,1,ACTIV !T0,2,ACTIV !T-1,2,ACTIV T-1,-1,INACT
Postconditions: P0,1 SACTIV A**1*

- **Rule**: North activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N0*1*: the rule concerns only the modules with a west neighbor, and without a north neighbor.
!E1,1: the grid cell in position $(1, 1)$ needs to be occupied.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A**1*: if the module was attached before and if still possible, it attaches to its new neighbors.

Rule Name: North for bridges Priority: 30 Preconditions: SBRIDG N011* T1,1,STOPP T-1,1,STOPP Postconditions: P0,1 SBRIDG A**1*

- **Rule**: North for bridges
- **Priority**: 30
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,1,STOPP: the module in position $(1, 1)$ needs to be set on *stop*.
T-1,1,STOPP: the module in position $(-1, 1)$ needs to be set on *stop*.
- **Postconditions**:
P0,1: the module moves north.
A**1*: the module attaches to its new east and west neighbors; if it was attached before and if still possible, it attaches to th other neighbors.

Rule Name: Stop bottommost module Priority: 9 Preconditions: N0100 !SBRIDG !SOBSTA !SSTOPP !SPATHS !SSTILL E-1,-1 !(!T-1,0,STOPP !T-1,0,PATHS !T-1,0,INACT) Postconditions: SSTOPP C006 + 0000 0001

- **Rule**: Stop bottommost module
- **Priority**: 9
- **Preconditions**:
!SBRIDG !SOBSTA !SSTILL !SSTOPP !SPATHS: it is not applicable to modules set in the *stop*, *bridge*, *still*, *obsta* nor *path* state.
N0100: the rule concerns only the modules with a west, and without other neighbors.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
!(!T-1,0,PATHS !T-1,0,INACT !T-1,0,STOPP): the cell grid in position $(-1, 0)$ needs to be occupied by a module set in the *inactive*, in the *stop* or in the *paths* state.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.
C006 + 0000 0001: the module sets its C06 counter to 1.

Rule Name: Stop other modules
Priority: 9
Preconditions: N0*01 !SBRIDG !SOBSTA !SSTILL !SSTOPP !SPATHS T0,-1,STOPP V0,-1,C006 C002 !V0,-1,C001 C001 W0,-1,C001 C001
Postconditions: SSTOPP C006 + 0,-1,C006 0001

- **Rule**: Stop other modules
- **Priority**: 9
- **Preconditions**:
!SBRIDG !SOBSTA !SSTILL !SSTOPP !SPATHS: it is not applicable to modules set in the *stop*, *bridge*, *still*, *obsta* nor *path* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north nor west neighbor.
T0,-1,STOPP: the module in position $(0, -1)$ needs to be set on *stop*.
(!V0,-1,C001 C001 W0,-1,C001 C001): the value of the counter C01 of the module has to coincide with the counter $C01$ of its south neighbor.
V0,-1,C006 C002: the value of the counter C02 of the module has to be bigger than the value of the counter C06 of its south neighbor.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.
C006 + 0,-1,C006 0001: the module checks the value of the counter C06 of its south neighbor, increases it by 1 and sets its own C06 counter to this result.

Rule Name: Bottommost module inactivation
Priority: 10
Preconditions: SSTOPP N*0*0 E-1,-1 E-1,1 !T0,1,ACTIV !T1,1,ACTIV !T-1,2,ACTIV !T2,1,ACTIV !T1,2,ACTIV
Postconditions: SINACT

- **Rule**: Bottommost module inactivation
- **Priority**: 10
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N*0*0: the rule concerns only the modules without a north nor a south neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule Name: Other modules inactivation
Priority: 10
Preconditions: N10*1 SSTOPP E-1,-1 !T0,1,ACTIV T0,-1,INACT E-1,0 E-1,1 !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV
Postconditions: SINACT

- **Rule**: Other modules inactivation
- **Priority**: 10
- **Preconditions**:

SSTOPP: it is applicable only to modules set in the *stop* state.
N10*1: the rule concerns only the modules with a north and a south neighbor, and without a west neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
T0,-1,INACT: the module in position $(0, -1)$ needs to be inactive.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
- Postconditions:
SINACT: the module changes its state to *inactive*.

Rule Name: Topmost module inactivation Priority: 12 Preconditions: SSTOPP N00*1 E-1,-1 T0,-1,INACT !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV !T1,2,ACTIV !T2,2,ACTIV Postconditions: SINACT

- Rule: Topmost module inactivation
- Priority: 12
- Preconditions:
SSTOPP: it is applicable only to modules set in the *stop* state.
N00*1: the rule concerns only the modules with a south neighbor, and without a north nor a west neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
T0,-1,INACT: the module in position $(0, -1)$ needs to be inactive.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2, 2)$ cannot be set on *active*.
- Postconditions:
SINACT: the module changes its state to *inactive*.

Rule Name: Locomotion break Priority: 100 Preconditions: SSTOPP N*010 = C001 0001 > C009 C010 Postconditions: SSTILL

- Rule: Locomotion break
- Priority: 100
- Preconditions:
SSTOPP: it is applicable only to modules set in the *stop* state.

N*010: the rule concerns only the modules with a west neighbor, and without a east nor south neighbor.
= C001 0001: the rule applies only if the value of the counter C01 of the module equals 1.
> C009 C010: the rule applies only if the value of the counter C09 is bigger than the value of C10.
- Postconditions:
SSTILL: the module changes its state to *still*.

Rule Name: Exchange rule for Bridges Priority: 100 Preconditions: SBRIDG !W0,1,C001 C001 !E0,1 = C007 0000 Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000 C017 + 0,1,C004 0000 C007 + 0000 0001

- Rule: Exchange rule for Bridges
- Priority: 100
- Preconditions:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
!W0,1,C001 C001: the value of the counter C01 of the module has to be less than the value of the counter C01 of its north neighbor.
= C007 0000: the rule applies only if the value of the counter $C07$ of the module is 0.
- Postconditions:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its north neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its north neighbor into its own counter C12.
C017 + 0,1,C004 0000: the module stores the value of the counter C04 of its north neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Exchange rule for Actives Priority: 100 Preconditions: T0,-1,BRIDG V0,-1,C001 C001 = C007 0000 Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000 C017 + 0,-1,C004 0000 C007 + 0000 0001

- Rule: Exchange rule for Actives
- Priority: 100
- Preconditions:
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.
V0,-1,C001 C001: the value of the counter C01 of the module has to be bigger than the value of the counter C01 of its south neighbor.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- Postconditions:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.

C017 + 0,-1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Internal change of counters I Priority: 100 Preconditions: SRINFO E0,1 = C007 0001 Postconditions: C001 + C011 0000 C002 + C012 0000 C004 + C017 0000 SACTIV

- **Rule**: Internal change of counters I
- **Priority**: 100
- **Preconditions**:
SRINFO: it is applicable only to modules set in the *rinfo* state. E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.
C004 + C017 0000: the module stores the value of the counter C17 of its south neighbor into its own counter C04.

Rule Name: Internal change of counters II Priority: 100 Preconditions: SRINFO !E0,1 = C007 0001 Postconditions: C001 + C011 0000 C002 + C012 0000 C004 + C017 0000 SBRIDG

- **Rule**: Internal change of counters II
- **Priority**: 100
- **Preconditions**:
SRINFO: it is applicable only to modules set in the *rinfo* state. !E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.
C004 + C017 0000: the module stores the value of the counter C17 of its south neighbor into its own counter C04.

Rule Name: Case of the last column (Bridge) Priority: 100 Preconditions: SBRIDG T0,1,ACTIV = C001 C013 = C007 0000 !(!V0,1,C001 C013 W0,1,C001 C013) Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000 C017+ 0,1,C004 0000 C007 + 0000 0001

- **Rule**: Case of the last column (Bridge)
- **Priority**: 100

- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
T0,1,ACTIV: the module in position $(0, 1)$ needs to be set on *active*.
!(!V0,1,C001 C013 W0,1,C001 C013): the value of the counter C01 of the module has to be different from the value of the counter C13 of its north neighbor.
= C001 C013: the rule applies only if the value of the counter C01 of the module is equal to the value of its counter C13.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C017 + 0,1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Case of the last column (Actives) Priority: 100 Preconditions: SACTIV T0,-1,BRIDG ¡ C001 C013 !V0,-1,C001 C013 W0,-1,C001 C013 = C007 0000 Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000 C017 + 0,-1,C004 0000 C007 + 0000 0001

- **Rule**: Case of the last column (Actives)
- **Priority**: 100
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.
(!V0,-1,C001 C013 W0,-1,C001 C013): the value of the counter C01 of the module has to be equal to the value of the counter C13 of its south neighbor.
< C001 C013: the rule applies only if the value of the counter C01 of the module is less than the value of its counter C13.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C017 + 0,-1,C004 0000: the module stores the value of the counter C04 of its south neighbor into its own counter C17.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Counter C007
Priority: 15
Preconditions: SBRIDG E0,1
Postconditions: C007 + 0000 0000 C017 + 0000 0000

- **Rule**: Counter C007
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
- **Postconditions**:
C017 + 0000 0000: the modules sets to 0 its counter C17.
C007 + 0000 0000: the modules sets to 0 its counter C07.

Rule Name: Counter C018
Priority: 150
Preconditions: SACTIV T0,-1,STOPP = C013 C001 !V0,-1,C001 0001 W0,-1,C001 0001 > C009 C010 = C019 0000
Postconditions: C018 + 0000 0001 C019 + 0000 0001

- **Rule**: Counter C018
- **Priority**: 150
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,STOPP: the grid cell in position $(0, -1)$ needs to be set in the *stop* state.
= C013 C001: the rule applies only if the value of the counter C01 meets the value of its counter C13.
> C009 C010: the rule applies only if the value of the counter C09 of the module is bigger than the value of its counter C10.
= C019 0000: the rule applies only if the value of the counter C19 is 0.
(!V0,-1,C001 0001 W0,-1,C001 0001): the value of the counter C01 of the module in position $(0, -1)$ has to be equal to 1.
- **Postconditions**:
C018 + 0000 0001: the modules sets to 1 its counter C18.
C019 + 0000 0001: the modules sets to 1 its counter C19.

Rule Name: Counting of the modules of the last column
Priority: 100
Preconditions: SBRIDG !E0,1 = C013 C001 = C016 0000 !V0,1,C018 0001 W0,1,C018 0001 > C009 C010
Postconditions: C014 + C014 0001 C016 + 0000 0001 C018 + 0000 0001

- **Rule**: Counting of the modules of the last column
- **Priority**: 100
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
= C013 C001: the rule applies only if the value of the counter C01 of the module equals the value of its counter C13.
= C016 0000: the rule applies only if the value of the counter C16 of the module is 0.
> C009 C010: the rule applies only if the value of the counter C09 of the module is bigger than the value of its counter C10.
(!V0,1,C018 0001 W0,1,C018 0001): the value of the counter C18 of the module in position $(0, 1)$ has to be equal 1.

- **Postconditions**:
C014 + C014 0001: the module adds 1 to its C04 counter.
C016 + 0000 0001: the module sets its C16 counter to 1.
C018 + 0000 0001: the modules sets to 1 its counter C18.

Rule Name: Auxiliar counter for the counting Priority: 100 Preconditions: SBRIDG E0,1 = C016 0001 Postconditions: C016 + 0000 0000

- **Rule**: Auxiliary counter for the counting
- **Priority**: 100
- **Preconditions**:
SBRIDGE: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C016 0001: the rule applies only if the value of the counter C16 of the module is 1.
- **Postconditions**:
C016 + 0000 0000: the module adds 0 to its C04 counter.

Rule Name: Bridges reconfiguration Priority: 100 Preconditions: SBRIDG E0,1 = C014 C015 Postconditions: SMOVES

- **Rule**: Bridges reconfiguration
- **Priority**: 100
- **Preconditions**:
SBRIDGE: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C014 C015: the rule applies only if the value of the counter C14 of the module equals the value of its counter C15.
- **Postconditions**:
SMOVES: the module changes its state to *moves*.

Rule Name: Path formation I Priority: 15 Preconditions: SACTIV T1,0,OBSTA Postconditions: SPATHS

- **Rule**: Path formation I
- **Priority**: 15
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T1,0,OBSTA: the grid cell in position $(1, 0)$ needs to set on *obsta*.
- **Postconditions**:
SPATHS: the module changes its state to *moves*.

Rule Name: Path formation II Priority: 14 Preconditions: SACTIV T0,-1,OBSTA Postconditions: SPATHS

- **Rule**: Path formation II

- **Priority**: 14
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,OBSTA: the grid cell in position $(1, 0)$ needs to set on *obsta*.
- **Postconditions**:
SPATHS: the module changes its state to *moves*.

Rule Name: Path formation III Priority: 12 Preconditions: SACTIV T-1,0,OBSTA !T1,0,STOPP Postconditions: SPATHS

- **Rule**: Path formation III
- **Priority**: 12
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T-1,0,OBSTA: the grid cell in position $(-1, 0)$ needs to set on *obsta*.
!T1,0,STOPP: the grid cell in position $(1, 0)$ cannot be set on *stop*.
- **Postconditions**:
SPATHS: the module changes its state to *moves*.

Rule Name: Path formation IV Priority: 13 Preconditions: SACTIV T-1,-1,OBSTA !T1,0,STOPP Postconditions: SPATHS

- **Rule**: Path formation IV
- **Priority**: 13
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T-1,-1,OBSTA: the grid cell in position $(-1, -1)$ needs to set on *obsta*.
!T1,0,STOPP: the grid cell in position $(1, 0)$ cannot be set on *stop*.
- **Postconditions**:
SPATHS: the module changes its state to *moves*.

Rule Name: Path formation V Priority: 11 Preconditions: SACTIV T1,-1,OBSTA Postconditions: SPATHS

- **Rule**: Path formation V
- **Priority**: 11
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
T1,-1,OBSTA: the grid cell in position $(1, -1)$ needs to set on *obsta*.
- **Postconditions**:
SPATHS: the module changes its state to *moves*.

Rule Name: South-west rule for the path formation Priority: 9 Preconditions: SACTIV N**00 T-1,0,PATHS E-1,-1 T-2,-1,OBSTA Postconditions: P-1,-1 SPATHS A11*1 C009 - C009 0001

- **Rule**: South-west rule for the path formation
- **Priority**: 9

- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N**00: the rule concerns only the modules without a east and a south neighbor.
T-1,0,PATHS: the grid cell in position $(-1, 0)$ needs to set on *paths*.
- **Postconditions**:
P-1,-1: the module moves south-west.
SPATHS: the module changes its state to *moves*.
A11*1: the module attaches to its north, east and south neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C009 - C009 0001: the module decreases the value of its C09 counter by 1.

Rule Name: North-west Inversion Priority: 100 Preconditions: SINACT N1000 T0,1,INACT E-1,1 Postconditions: P-1,1 SACTIV A**1* C009 - C009 0001

- **Rule**: North-west Inversion
- **Priority**: 100
- **Preconditions**:
SINACT: it is applicable only to modules set in the *inactive* state.
N1000: the rule concerns only the modules with a north neighbor and without any other neighbor.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
T0,1,PATHS: the grid cell in position $(0, 1)$ needs to set on *paths*.
- **Postconditions**:
P-1,1: the module moves north-west.
SACTIV: the module changes its state to *active*.
A**1*: the module attaches to its east neighbor; if it was attached before and if still possible, it attaches to the other neighbor.
C009 - C009 0001: the module decreases the value of its C09 counter by 1.

Rule Name: North-east Inversion Priority: 100 Preconditions: SACTIV N001* T1,0,INACT T2,0,PATHS E1,1 Postconditions: P1,1 A1*11 C009 + C009 0001

- **Rule**: North-east Inversion
- **Priority**: 100
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N001*: the rule concerns only the modules with a north neighbor and without any other neighbor.
E1,1: the grid cell in position $(1, 1)$ needs to be empty.
T1,0,INACT: the grid cell in position $(1, 0)$ needs to set on *inactive*.
T2,0,PATHS: the grid cell in position $(2, 0)$ needs to set on *paths*.
- **Postconditions**:
P1,1: the module moves north-east.
A1*11: the module attaches to its north, west and south neighbor; if it was attached before and if still possible, it attaches to the other neighbor.
C009 + C009 0001: the module increases the value of its C09 counter by 1.

Rule Name: Path reactivation I
Priority: 100
Preconditions: SPATHS N101* T0,1,PATHS T1,0,OBSTA !(!T0,-1,OBSTA !E0,-1) E-1,1 E-1,2
Postconditions: SACTIV P-1,1 C009 - C009 0001

- **Rule**: Path reactivation I
- **Priority**: 100
- **Preconditions**:
SPATHS: it is applicable only to modules set in the *paths* state.
N101*: the rule concerns only the modules with a north and a east neighbor, and without a west neighbor.
T0,1,PATHS: the grid cell in position $(0, 1)$ needs to set on *paths*.
T1,0,OBSTA: the grid cell in position $(1, 0)$ needs to set on *obsta*.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
E-1,2: the grid cell in position $(-1, 2)$ needs to be empty.
!(!T0,-1,OBSTA !E0,-1): the grid cell in position $(0, -1)$ has to be either occupied by an obstacle module, either empty.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P-1,1: the module moves north-east.
C009 - C009 0001: the module decreases the value of its C09 counter by 1.

Rule Name: Path reactivation II
Priority: 100
Preconditions: SPATHS N0*1* !(!T-1,0,OBSTA !E-1,0) !(!T0,-1,OBSTA !E0,-1) T1,0,PATHS E1,1 !T2,1,ACTIV
Postconditions: SACTIV P1,1 C009 + C009 0001

- **Rule**: Path reactivation II
- **Priority**: 100
- **Preconditions**:
SPATHS: it is applicable only to modules set in the *paths* state.
N0*1*: the rule concerns only the modules without a north neighbor, but with a east one.
E1,1: the grid cell in position $(1, 1)$ needs to be empty.
T1,0,PATHS: the grid cell in position $(1, 0)$ needs to set on *paths*.
!T2,1,ACTIV: the grid cell in position $(2, 1)$ needs to set on *active*.
!(!T0,-1,OBSTA !E0,-1): the grid cell in position $(0, -1)$ has to be either occupied by an obstacle module, either empty.
!(!T-1,0,OBSTA !E-1,0): the grid cell in position $(-1, 0)$ has to be either occupied by an obstacle module, either empty.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P1,1: the module moves north-east.
C009 + C009 0001: the module increases the value of its C09 counter by 1.

Rule Name: Path reactivation III
Priority: 100
Preconditions: SPATHS N0*01 T0,-1,PATHS !(!E-1,0 !T-1,0,OBSTA) E1,-1 !T1,-2,ACTIV !T1,-2,RINFO
Postconditions: SACTIV P1,-1 C009 + C009 0001

- **Rule**: Path reactivation III
- **Priority**: 100
- **Preconditions**:

SPATHS: it is applicable only to modules set in the *paths* state.
N0*01: the rule concerns only the modules without a north and a east neighbor, but with a south one.
E1,-1: the grid cell in position $(1, -1)$ needs to be empty.
T0,-1,PATHS: the grid cell in position $(0, -1)$ needs to set on *paths*.
!T1,-2,ACTIV !T1,-2,RINFO: the grid cell in position $(1, -2)$ needs to set either on the *active* or on the *rinfo* state.
!(!T-1,0,OBSTA !E-1,0): the grid cell in position $(-1, 0)$ has to be either occupied by an obstacle module, either empty.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P1,-1: the module moves north-west.
C009 + C009 0001: the module increases the value of its C09 counter by 1.

Rule Name: Path reactivation IV Priority: 100 Preconditions: SPATHS N0*01 T0,-1,PATHS !(!E-1,0 !T-1,0,OBSTA) !(!T1,-1,STOPP !T1,-1,PATHS) Postconditions: SACTIV P1,0 C009 + C009 0001

- **Rule**: Path reactivation IV
- **Priority**: 100
- **Preconditions**:
SPATHS: it is applicable only to modules set in the *paths* state.
N0*01: the rule concerns only the modules without a north and a east neighbor, but with a south one.
T0,-1,PATHS: the grid cell in position $(0, -1)$ needs to set on *paths*.
!(!T-1,0,OBSTA !E-1,0): the grid cell in position $(-1, 0)$ has to be either occupied by an obstacle module, either empty.
!(!T1,-1,STOPP !T1,-1,PATHS): the grid cell in position $(1, -1)$ has to be occupied by a stop or by a path module.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P1,0: the module moves east.
C009 + C009 0001: the module increases the value of its C09 counter by 1.

Rule Name: Path reactivation V Priority: 100 Preconditions: SPATHS N0*1* !(T0,-1,PATHS T1,0,PATHS) !(!T1,0,STOPP !T1,0,PATHS) !(!T1,1,STOPP !T1,1,PATHS) !(!T0,-1,OBSTA !T0,-1,PATHS !E0,-1) !(!E-1,0 !T-1,0,OBSTA) Postconditions: SACTIV P0,1

- **Rule**: Path reactivation V
- **Priority**: 100
- **Preconditions**:
SPATHS: it is applicable only to modules set in the *paths* state.
N0*1*: the rule concerns only the modules without a north neighbor, but with a east one.
T0,-1,PATHS: the grid cell in position $(0, -1)$ needs to set on *paths*.
!(!T-1,0,OBSTA !E-1,0): the grid cell in position $(-1, 0)$ has to be either occupied by an obstacle module, either empty.
!(!T1,1,STOPP !T1,1,PATHS): the grid cell in position $(1, 1)$ has to be occupied by a stop or by a path module.

!(T0,-1,PATHS T1,0,PATHS): the rule is not applicable if the grid cells $(0,-1)$ and $(1,0)$ are both occupied by path modules.
!(!T1,0,STOPP !T1,0,PATHS): the grid cell in position $(1,0)$ has to be occupied by a stop or by a path module.
!(!T0,-1,OBSTA !T0,-1,PATHS !E0,-1): the grid cell in position $(-1,0)$ has to be either occupied by an obstacle module, either by a path module, either empty.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P0,1: the module moves north.

Rule Name: Path reactivation VI Priority: 100 Preconditions: SPATHS N0*1* !(!E-1,0 !T-1,0,OBSTA) T1,0,STOPP E1,1 Postconditions: SACTIV P1,1 C009 + C009 0001

- **Rule**: Path reactivation VI
- **Priority**: 100
- **Preconditions**:
SPATHS: it is applicable only to modules set in the *paths* state.
N0*1*: the rule concerns only the modules without a north neighbor, but with a east one.
E1,1: the grid cell in position $(1,1)$ needs to be empty.
T1,0,STOPP: the grid cell in position $(1,0)$ needs to set on *stop*.
!(!T-1,0,OBSTA !E-1,0): the grid cell in position $(-1,0)$ has to be either occupied by an obstacle module, either empty.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
P1,1: the module moves north-east.
C009 + C009 0001: the module increases the value of its C09 counter by 1.

South (Case of one column) 25 !SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS N*100 !E-1,-1 !T0,-2,ACTIV !(T1,-2,ACTIV T0,-2,STOPP) !T0,-2,RINFO !T1,-2,RINFO !T-1,-1,RINFO !(T0,-2,BRIDG T1,-1,STOPP) !T1,-2,ACTIV !(T-1,0,PATHS T-1,-1,OBSTA) !T1,-1,ACTIV P0,-1 SACTIV A***1 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: South (Case of one column)
- **Priority**: 25
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA !SPATHS: it is not applicable to modules set in the *stop*, *bridge*, *still*, *obsta* nor *path* state.
N*100: the rule concerns only the modules with a west neighbor, and without a east or a south neighbor.
!E-1,-1: the cell grid in position $(-1,-1)$ needs to be occupied.
!T1,-1,ACTIV: the module in position $(1,-1)$ cannot be set on *active*.
!T0,-2,ACTIV: the module in position $(0,-2)$ cannot be set on *active*.
!T1,-2,ACTIV: the module in position $(1,-2)$ cannot be set on *active*.
!T1,-2,RINFO: the module in position $(1,-2)$ cannot be set on *rinfo*.
!T-1,-1,RINFO: the module in position $(-1,-1)$ cannot be set on *rinfo*.
!T0,-2,RINFO: the module in position $(0,-2)$ cannot be set on *rinfo*.

!(T1,-2,ACTIV T0,-2,STOPP): the rule cannot be applied if the module $(1, -2)$ is set on *active* and the module $(0, -2)$ is set on *stop*.
!(T0,-2,BRIDG T1,-1,STOPP): the rule cannot be applied if the module $(0, -2)$ is set on *bridge* and the module $(1, -1)$ is set on *stop*.
!(T-1,0,PATHS T-1,-1,OBSTA): the rule cannot be applied if the module $(-1, 0)$ is set on *paths* and the module $(-1, -1)$ is set on *obsta*.
- Postconditions:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

```
South for the first module of the path (Case of one column)
30
SACTIV N0110 T1,0,OBSTA T1,-1,OBSTA !(!T-1,-1,INACT !T-1,-1,STOPP)
P0,-1 A*11* C007 + 0000 0000 C017 + 0000 0000
```

- Rule: South for the first module of the path (Case of one column)
- Priority: 30
- Preconditions:
SACTIV: it is applicable to modules set in the *active* state.
N0110: the rule concerns only the modules with a west and a east neighbor, and without a north and a south neighbor.
T1,0,OBSTA: the module in position $(1, 0)$ needs to be an *obstacle*.
T1,-1,OBSTA: the module in position $(1, -1)$ needs to be an *obstacle*.
!(!T-1,-1,INACT !T-1,-1,STOPP): the module in position $(-1, -1)$ needs to be set either in the *inactive* or in the *stop* state.
- Postconditions:
P0,-1: the module moves south.
A*11*: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

```
First module of the path (Case of one column)
15
SACTIV N0110 E-1,-1 T1,0,OBSTA
SPATHS
```

- Rule: First module of the path (Case of one column)
- Priority: 15
- Preconditions:
SACTIV: it is applicable to modules set in the *active* state.
N0110: the rule concerns only the modules with a west and a east neighbor, and without a north and a south neighbor.
T1,0,OBSTA: the module in position $(1, 0)$ needs to be an *obstacle*.
E-1,-1: the cell grid in position $(-1, -1)$ needs to be empty.
- Postconditions:
SPATHS: the module changes its state to *paths*.

```
First module of the path (Case of one column)
15
SACTIV N0110 E-1,-1 T1,0,OBSTA
SPATHS
```

- **Rule**: Path formation I (Case of one column)
- **Priority**: 15
- **Preconditions**:

SACTIV: it is applicable to modules set in the *active* state.
N0110: the rule concerns only the modules with a west and a east neighbor, and without a north and a south neighbor.
T1,0,OBSTA: the module in position $(1, 0)$ needs to be an *obstacle*.
T0,-1,PATHS: the module in position $(0, -1)$ needs to be set on *paths*.

- **Postconditions**:

SPATHS: the module changes its state to *paths*.

Chapter 9

Histogram locomotion under superior obstacles

9.1 Goal

In this chapter we discuss the eastward locomotion of a modular robotic system initially configured as an histogram, in the presence of superior obstacles. As in the free locomotion case, the rules are divided into locomotion and reconfiguration rules: as soon as the system crosses the obstacle, it reconfigures into its initial shape.

9.2 Strategy

The crossing of superior obstacles by an histogram presents many analogies with the case of the rectangle; as in the rectangle case, in fact, the strategy of the free locomotion is conserved, but with some modifications that make it possible to crawl under the obstacles.

Before encountering the obstacles, the modules of the leftmost column in turn move from the back of the group, over the top, and locate on the front of the group to reform a new column with the same height. As soon as the presence of an obstacle is detected, the system continues its locomotion dividing the columns into different subcolumns when it is necessary, flattening in order to be able to crawl under the obstacle and reach its end. While walking under the obstacle then, columns are reconfigured entirely only when possible, and divided into different parts if the total configuration is not achievable due to the height restrictions produced by the obstacle.

We can observe an example of the crossing of a superior obstacle in Figure 9.1. As the system has crossed the obstacle, it reconfigures into the initial shape and stops the locomotion.

As in the case of the rectangle, we can observe how this set of rules produce both the crossing of superior obstacles configured as histograms, and the crossing of superior general obstacles; in fact, the system does not detect any difference between the two cases, as it just moves taking into account the presence of obstacle modules without using, nor explicitly nor in an implicit way, the fact

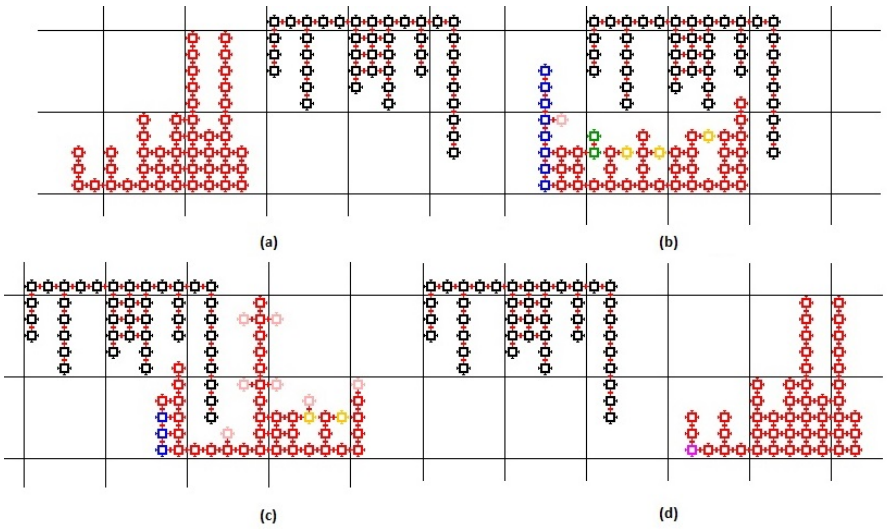


Figure 9.1: Crawling under a superior histogram obstacle: in (a) we can see that the system is configured as an histogram when it starts the locomotion. As the obstacle is detected in (b) and (c), as the locomotion continues, the columns are sometimes not configured completely, but divided into subcolumns: the system flattens and decreases its height in order to be able to cross. After the obstacle, the system reconfigures again and stops the locomotion in (d). Stopped modules are red, active modules are pink, inactive modules are blue, obstacles are black and green modules are exchanging information.

that they belong to an histogram. In Figure 9.2 we can see an example of the crossing of a general obstacle.

9.3 Locomotion

The set of rules presented in this chapter is a direct adaptation of the set of rules that produces the free locomotion of histograms presented in Chapter 7; the number of the rules, the structure and the purpose of each one of them is the same as before, apart from some adaptations that make the system able to continue the locomotion in the presence of superior obstacles.

In this section we present and explain a few examples of such modifications through the analysis of the new East rule and the new Stop bottommost module rule, and then we skip directly to the study of the complexity of the rules and to their detailed presentation in sections 9.5 and 9.6, as no one of these modifications compromises the correctness, and the results proved in Chapter 7 are still valid and can be proved in the same way.

In Figure 9.3 we can see the details of the new version of the East rule; the differences introduced are due to the possible presence of obstacles over the system.

In Figure 9.4 the "Stop other modules" rule is presented; this is one of the

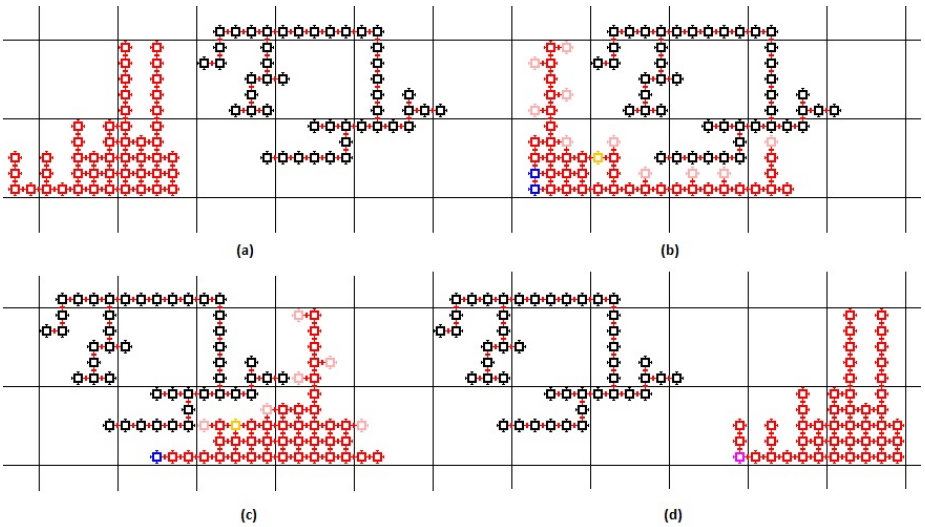


Figure 9.2: Crawling under a superior general obstacle: the system behaves as in the previous cases. Stopped modules are red, active modules are pink, inactive modules are blue and obstacles are black.

most important modification, as it allows the flattening of the system and the reconfiguration of columns into different subcolumns while the system is crawling under the obstacle.

9.4 Reconfiguration

We recall that the reconfiguration strategy of the free locomotion case is based on the counting of the rounds of the system; such rounds are controlled by the counter C04, which counts the number of times each module changes its state from stop to inactive, i.e. each time its column starts the movement again. In the case of the free locomotion, this counter is indicative for the rounds of the system, as modules of the same column inactivate together and stop together every time the column moves, and no differences can be produced between modules of the same column. In the case of superior obstacles, however, it can happen that modules of the same column inactivate a different number of times during the locomotion, due to the divisions into subcolumns operated by the system while crossing the obstacle; we can see an example of such case in Figure 9.5. Because of this reason, the reconfiguration of the system is done through the strategy applied in the case of inferior obstacles, presented in Chapter 8.

East
Priority: 7
Preconditions: !SSTOPP !SSTILL !SOBSTA !SBRIDG N**01 !E1,-1 !T1,-1,MOVES !T1,-1,ACTIV !T1,1,ACTIV !T2,0,ACTIV !T2,1,ACTIV !T2,-1,ACTIV !T2,0,RINFO !T2,-1,RINFO !T1,-1,RINFO !(SINACT T1,-1,BRIDG) !(!T0,1,OBSTA !E0,1)
Postconditions: SACTIV P1,0 A**11 C007 + 0000 0000 C017 + 0000 0000

Figure 9.3: Details of the East rule: the new condition *!SOBSTA* (state $\neq$ *obstacle*) is introduced, as in all the other rules, in order to deal with the introduction of a new state for the modules; moreover, the previous condition *N0*01* is replaced by *N**01 !(!T0,1,OBSTA !E0,1)*, through which we allow the rule to be applied by a module with an obstacle as a north neighbor, preventing modules to stop if they detect an obstacle in the relative position $(0, 1)$.

Stop other modules
Priority: 9
Preconditions: SACTIV N0*01 T0,-1,STOPP !T-1,1,OBSTA !T1,1,OBSTA !V0,-1,C001 C001 W0,-1,C001 C001 V0,-1,C006 C002
Postconditions: SSTOPP C006 + 0,-1,C006 0001

Figure 9.4: Details of the *Stop other modules* rule: *!T-1,1,OBSTA !T1,1,OBSTA* in a new condition that allows the stop of the module only in the cases in which this would not imply a block for the passage of the other modules; in case the module cannot apply this rule, it would continue the movement, reach the bottommost cell on the right of the system and stop through the bottommost module rule, continuing the reconfiguration of its column into a new subcolumn and allowing the system to continue the locomotion.

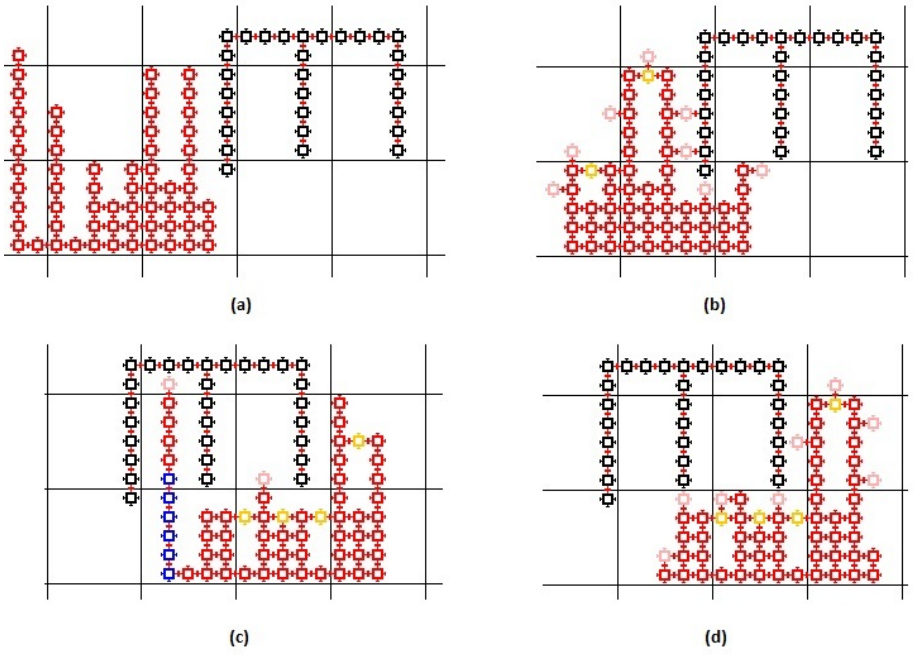


Figure 9.5: Example of a situation that produces a difference in the counter C04 among modules of the same column: we can see in (a) that column 1 is initially formed by 11 modules; as they reach the right of the system in (b), they reconfigure into 3 subcolumns (the three rightmost columns in (b)). As soon as the first two change their state to inactive again, they reach the last one of the three, stop again and reconfigure the entire column in (c); they inactivate again in (d), continuing the locomotion. In this process, the first modules of the column have increased their C04 by two, while the rest of the column have inactivated only once.

9.5 Complexity

Neighbourhood As in the case of the free locomotion, each module is able to check if the grid cells of its first and its second neighborhood are either empty or occupied by another module; in this last case, it has to be able to obtain information about the state of such module and about the value of its counters. The checking of the second neighborhood modules is needed in order to avoid collisions between active modules and moving bridges: as a bridge moves, its neighborhood needs to be free of active modules willing to move to its goal grid cell; this is achieved by the generation of a delay between two consecutive active modules. No information about any other position is needed. No information about any other position is needed.

Memory and computation The application of this set of rules requires $O(1)$ memory for each module, and a $O(1)$ computation at each step, as modules only need to check if preconditions of rules are fulfilled, to memorize fixed values in some counters, and to make simple operations with some counter at each step.

Number of moves We analyze here the number of moves performed by a module from the moment it activates to its inactivation; as we have already analyzed the case of free locomotion, we focus now on the crossing of the obstacle. Starting from the worst case analysis presented for the case of the free locomotion of the histogram in Chapter 7, we want to understand now how the presence of a superior obstacle could affect the maximum number of moves of a module. Given the strategy of the movement, the effect of the presence of such obstacle is, just as in the case of the rectangle, the reconfiguration of the columns into more different subcolumns. Supposing that the histogram starts its movement as in the configuration depicted in Figure 7.11 of Chapter 7, which represents the worst case for the number of moves in the free locomotion, the effect of the presence of a superior obstacle would be the subdivision of the columns of maximal height h into different consecutive subcolumns. This would actually produce an improvement in the number of time steps of a module, as the number of modules that need to be touched more than one times during the locomotion would decrease (there would be less modules with both sides free). This means that the worst case possible for the number of moves in the presence of superior obstacles is the one in which the histogram is configured in the described way, and the superior obstacle is as far from the ground as the system does not perceive it: in this case a module performs $O(n)$ moves. The overall number of moves for a module during the overpassing of the obstacle is then $O(kn)$ at most, where k is the number of rounds of the system, which depends on the width of the obstacle.

Communication As no additional exchange of communication is performed during the crossing of the obstacle, the communication operated by a module is at most $O(n)$ at each round, as in the case of the free locomotion. The need of information about the second neighborhood is the need to

avoid collisions between the moving bridges and the active modules, as in the case of free locomotion.

Number of time steps The computation of the number of time steps is analogous to the one presented in the case of free locomotion: as the effect of the presence of superior obstacle is the subdivision of the columns into different subcolumns, the computation is the same as in the free locomotion, only done considering a higher number of columns and with less modules for each column considered. The overpassing of a superior obstacle can then be performed in $O(kn)$ time steps, with k the number of rounds performed (linearly dependent on the number of modules of the system) and n is the overall number of modules of the system.

9.6 Rules

The algorithm is based on 26 different rules; before starting the locomotion each module has the following information stored in its counters: the number of its the column in the order defined (C01), the height of its column (C02), the number of the columns of the histogram (C13), the height of the last column (C15) and the number of rounds (C10).

Rule Name: North
Priority: 6
Preconditions: !SSTOPP !SBRIDG !SSTILL !SOBSTA N0*1* T1,1,STOPP !T0,2,ACTIV !T1,2,ACTIV
Postconditions: P0,1 SACTIV A1*1* C007 + 0000 0000 C017 + 0000 0000

- **Rule**: North
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBTA: it is not applicable to modules set in the *stop*, *bridge*, *still* nor *obstacle* state.
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
T1,1,STOPP: the module in position $(1, 1)$ needs to be set on *stop*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A1*1*: the module attaches to its new north and east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

Rule Name: North-east
Priority: 6
Preconditions: !SSTOPP !SSTILL !SBRIDG !SOBSTA N0*1* E1,1 T1,0,STOPP !T2,1,RINFO !T2,0,RINFO !T2,1,ACTIV !T1,2,ACTIV !T2,2,ACTIV
Postconditions: SACTIV P1,1 A***1 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001

- **Rule**: North-east
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA: it is not applicable to modules set in the *stop*, *bridge*, *still* nor *obstacle* state.
N0*1*: the rule concerns only the modules with a east neighbor, and without a north neighbor.
E1,1: the grid cell in position $(1, 1)$ needs to be empty.
T1,0,STOPP: the module in position $(1, 0)$ needs to be set on *stop*.
!T2,1,RINFO: the module in position $(2, 1)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
P1,1: the module moves north-east.

SACTIV: the module changes its state to *active*.
A***1: the module attaches to its new south neighbor; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: East Priority: 7 Preconditions: !SSTOPP !SSTILL !SBRIDG !SOBSTA N**01 !(!T0,1,OBSTA !E0,1) !E1,-1 !T1,-1,MOVES !T1,-1,ACTIV !T1,1,ACTIV !T2,0,ACTIV !T2,1,ACTIV !T2,-1,ACTIV !T2,0,RINFO !T2,-1,RINFO !T1,-1,RINFO !(SINACT T1,-1,BRIDG) Postconditions: SACTIV P1,0 A**11 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001

- **Rule**: East
- **Priority**: 7
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA: it is not applicable to modules set in the *stop*, *bridge*, *still* or *obstacle* state.
N**01: the rule concerns only the modules with a south neighbor, and without a north nor a east neighbor.
!(!T0,1,OBSTA !E0,1): the cell grid in position $(0, 1)$ needs to be either empty or occupied by and obstacle module. !E1,-1: the cell grid in position $(1, -1)$ needs to be occupied.
!T1,-1,MOVES: the module in position $(1, -1)$ cannot be set on *moves*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(2, -1)$ cannot be set on *active*.
!T2,0,RINFO: the module in position $(2, 0)$ cannot be set on *rinfo*.
!T2,-1,RINFO: the module in position $(2, -1)$ cannot be set on *rinfo*.
!T1,-1,RINFO: the module in position $(1, -1)$ cannot be set on *rinfo*.
!(SINACT T1,-1,BRIDG): the rule is not applicable by an inactive module if the position $(1, -1)$ is occupied by a bridge.
- **Postconditions**:
P1,0: the module moves east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: South Priority: 6 Preconditions: !SSTOPP !SBRIDG !SSTILL !SOBSTA N*1*0 !(!T1,0,OBSTA !E1,0) !E-1,-1 !T0,-2,ACTIV !T0,-2,MOVES !(T1,-2,ACTIV T0,-2,STOPP) !T0,-2,RINFO !T1,-2,RINFO !(T0,-2,BRIDG T1,-1,STOPP) !T1,-1,ACTIV Postconditions: P0,-1 SACTIV A*111 C007 + 0000 0000 C017 + 0000 0000

- **Rule**: South
- **Priority**: 6

- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA: it is not applicable to modules set in the *stop*, *bridge*, *still* or *obstacle* state.
N*1*0: the rule concerns only the modules with a west neighbor, and without a south neighbor.
!(!T1,0,OBSTA !E1,0): the grid cell in position (1,0) needs to be either empty or occupied by an obstacle module.
!E-1,-1: the cell grid in position $(-1,-1)$ needs to be occupied.
!T0,-2,ACTIV: the module in position $(0,-2)$ cannot be set on *active*.
!T0,-2,MOVES: the module in position $(0,-2)$ cannot be set on *moves*.
!T0,-2,RINFO: the module in position $(0,-2)$ cannot be set on *rinfo*.
!T1,-2,RINFO: the module in position $(1,-2)$ cannot be set on *rinfo*.
!T1,-1,ACTIV: the module in position $(1,-1)$ cannot be set on *active*.
!(T1,-2,ACTIV T0,-2,STOPP): if the module in position $(1,-2)$ is set on *active* and the module in position $(0,-2)$ is set on *stop* the rule cannot be performed.
!(T0,-2,BRIDG T1,-1,STOPP): if the module in position $(0,-2)$ is set on *bridge* and the module in position $(1,-1)$ is set on *stop* the rule cannot be performed
- **Postconditions**:
P0,-1: the module moves south.
SACTIV: the module changes its state to *active*.
A*111: the module attaches to its new south and east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.

```
Rule Name: South-east
Priority: 6
Preconditions: !SSTOPP !SBRIDG !SSTILL !SOBSTA N**01 !(!E0,1 !T0,1,OBSTA) E1,-1
!T0,-1,ACTIV !T1,-2,ACTIV !T2,0,ACTIV !T1,-2,RINFO !T2,0,RINFO
!T2,-1,ACTIV !T1,-2,MOVES !(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP)
Postconditions: P1,-1 SACTIV A*111 C007 + 0000 0000 C017 + 0000 0000 C009 + C009 0001
```

- **Rule**: South-east
- **Priority**: 6
- **Preconditions**:
!SSTOPP !SBRIDG !SSTILL !SOBSTA: it is not applicable to modules set in the *stop*, *bridge*, *still* or *obstacle* state.
N**01: the rule concerns only the modules with a south neighbor, and without a north nor a east neighbor.
!(!E0,1 !T0,1,OBSTA): the grid cell in position (0,1) needs to be either empty or occupied by an obstacle module.
E1,-1: the cell grid in position $(1,-1)$ needs to be empty.
!T0,-1,BRIDG: the module in position $(0,-1)$ cannot be set on *bridge*.
!T0,-1,ACTIV: the module in position $(0,-1)$ cannot be set on *active*.
!T1,-2,ACTIV: the module in position $(1,-2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(2,-1)$ cannot be set on *active*.
!T1,-2,RINFO: the module in position $(1,-2)$ cannot be set on *rinfo*.
!T2,0,RINFO: the module in position $(2,0)$ cannot be set on *rinfo*.
!T1,-2,MOVES: the module in position $(1,-2)$ cannot be set on *moves*.

!(T1,-2,BRIDG T2,-1,STOPP T0,-1,STOPP): if the module in position $(1, -2)$ is set on *bridge* and the modules in position $(2, -1)$ and $(0, -1)$ are set on *stop* the rule cannot be performed.
- **Postconditions**:
P1,-1: the module moves south-east.
SACTIV: the module changes its state to *active*.
A*111: the module attaches to its new south and east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbor.
C007 + 0000 0000: the module sets to zero its C07 counter.
C017 + 0000 0000: the module sets to zero its C07 counter.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: Bridge Formation
Priority: 10
Preconditions: SACTIV N011* T1,0,STOPP T-1,0,STOPP
Postconditions: SBRIDG

- **Rule**: Bridge formation
- **Priority**: 10
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,0,STOPP: the module in position $(1, 0)$ needs to be set on *stop*.
T-1,0,STOPP: the module in position $(-1, 0)$ needs to be set on *stop*.
- **Postconditions**:
SBRIDG: the module changes its state to *bridge*.

Rule Name: North-east activation for bridges
Priority: 15
Preconditions: SBRIDG N011* E1,1 !T-1,1,ACTIV !T-1,2,ACTIV !T0,2,ACTIV !T1,0,ACTIV T-1,-1,INACT
Postconditions: P1,1 SACTIV A**11 C009 + C009 0001

- **Rule**: North-east activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
E1,1: the cell grid in position $(1, 1)$ needs to be empty.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P1,1: the module moves north-east.
SACTIV: the module changes its state to *active*.
A**11: the module attaches to its new south and east neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C009 + C009 0001: the module increases its C09 value by one.

Rule Name: North activation for bridges
Priority: 15
Preconditions: SBRIDG N011* !E1,1 !T-1,1,ACTIV !T1,1,ACTIV !T0,2,ACTIV
!T-1,2,ACTIV T-1,-1,INACT
Postconditions: P0,1 SACTIV A**1*

- **Rule**: North activation for bridges
- **Priority**: 15
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
!E1,1: the grid cell in position $(1, 1)$ needs to be occupied.
!T-1,1,ACTIV: the module in position $(-1, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(0, 2)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
T-1,-1,INACT: the module in position $(-1, -1)$ needs to be set on *inactive*.
- **Postconditions**:
P0,1: the module moves north.
SACTIV: the module changes its state to *active*.
A**1*: the module attaches to its new east neighbor; if it was attached before and if still possible, it attaches to the other neighbors.

Rule Name: North for bridges
Priority: 30
Preconditions: SBRIDG N011* T1,1,STOPP T-1,1,STOPP
Postconditions: P0,1 A**1*

- **Rule**: North for bridges
- **Priority**: 30
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
N011*: the rule concerns only the modules with a west and east neighbor, and without a north neighbor.
T1,1,STOPP: the module in position $(1, 1)$ needs to be set on *stop*.
T-1,1,STOPP: the module in position $(-1, 1)$ needs to be set on *stop*.
- **Postconditions**:
P0,1: the module moves north.
A**1*: the module attaches to its new east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.

Rule Name: Stop bottommost module
Priority: 9
Preconditions: SACTIV N0100 !(!T-1,0,INACT !T-1,0,STOPP) E-1,-1
Postconditions: SSTOPP C006 + 0000 0001

- **Rule**: Stop bottommost module
- **Priority**: 9
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N0100: the rule concerns only the modules with a west, and without other neighbors.

E-1,-1: the grid cell in position $(-1,-1)$ needs to be empty.
!(!T-1,0,INACT !T-1,0,STOPP): the cell grid in position $(-1,0)$ needs to be occupied by a module, either set in the *inactive*either in the *stop* state.
- **Postconditions**:
P0,1: the module moves north.
SSTOPP: the module changes its state to *stop*.
A**1*: the module attaches to its new east and west neighbors; if it was attached before and if still possible, it attaches to the other neighbors.
C006 + 0000 0001: the module sets its C06 counter to 1.

Rule Name: Stop other modules Priority: 9 Preconditions: SACTIV N0*01 T0,-1,STOPP !T-1,1,OBSTA !T1,1,OBSTA !V0,-1,C001 C001 W0,-1,C001 C001 V0,-1,C006 C002 Postconditions: SSTOPP C006 + 0,-1,C006 0001

- **Rule**: Stop other modules
- **Priority**: 9
- **Preconditions**:
SACTIV: it is applicable only to modules set in the *active* state.
N0*01: the rule concerns only the modules with a south neighbor, and without a north nor west neighbor.
T0,-1,STOP: the module in position $(0,-1)$ needs to be set on *stop*.
!T-1,1,OBSTA: the module in position $(-1,1)$ cannot be set on *obstacle*.
!T1,1,OBSTA: the module in position $(1,1)$ cannot be set on *obstacle*.
(!V0,-1,C001 C001 W0,-1,C001 C001): the value of the counter C01 of the module has to coincide with the counter $C01$ of its south neighbor.
V0,-1,C006 C002: the value of the counter C02 of the module has to be bigger than the value of the counter C06 of its south neighbor.
- **Postconditions**:
SSTOPP: the module changes its state to *stop*.
C006 + 0,-1,C006 0001: the module checks the value of the counter C06 of its south neighbor, increases it by 1 and sets its own C06 counter to this result.

Rule Name: Locomotion break Priority: 100 Preconditions: SSTOPP N*0*0 = C001 0001 > C009 C010 Postconditions: SSTILL

- **Rule**: Locomotion break
- **Priority**: 100
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N*010: the rule concerns only the modules with a west neighbor, and without a east nor south neighbor.
= C001 0001: the rule applies only if the value of the counter C01 of the module equals 1.
> C009 C010: the rule applies only if the value of the counter C09 is bigger than the value of C10.
- **Postconditions**:
SSTILL: the module changes its state to *still*.

Rule Name: Bottommost module inactivation
Priority: 10
Preconditions: SSTOPP N*0*0 E-1,-1 E-1,1 !T0,1,ACTIV !T1,1,ACTIV !T-1,2,ACTIV !T2,1,ACTIV !T1,2,ACTIV
Postconditions: SINACT C004 + C004 0001

- **Rule**: Bottommost module inactivation
- **Priority**: 10
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N*0*0: the rule concerns only the modules without a west or a south neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
!T0,1,ACTIV: the module in position $(0, 1)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T-1,2,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(1, 2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule Name: Other modules inactivation
Priority: 10
Preconditions: SSTOPP N10*1 E-1,-1 T0,-1,INACT E-1,0 E-1,1 !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV
Postconditions: SINACT

- **Rule**: Other modules inactivation
- **Priority**: 10
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N10*1: the rule concerns only the modules with a north and a south neighbor, and without a west neighbor.
E-1,-1: the grid cell in position $(-1, -1)$ needs to be empty.
E-1,1: the grid cell in position $(-1, 1)$ needs to be empty.
T0,-1,INACT: the module in position $(0, -1)$ needs to be inactive.
!T1,0,ACTIV: the module in position $(1, 0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1, 1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1, -1)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(2, 1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1, 2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2, 0)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2, 2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule Name: Other modules inactivation
Priority: 10
Preconditions: SSTOPP N10*1 E-1,-1 T0,-1,INACT E-1,0 E-1,1 !T1,0,ACTIV !T1,1,ACTIV !T1,-1,ACTIV !T0,2,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,1,ACTIV
Postconditions: SINACT

- **Rule**: Topmost module inactivation
- **Priority**: 12
- **Preconditions**:
SSTOPP: it is applicable only to modules set in the *stop* state.
N00*1: the rule concerns only the modules with a south neighbor, and without a north nor a west neighbor.
E-1,-1: the grid cell in position $(-1,-1)$ needs to be empty.
T0,-1,INACT: the module in position $(0,-1)$ needs to be inactive.
!T1,0,ACTIV: the module in position $(1,0)$ cannot be set on *active*.
!T1,1,ACTIV: the module in position $(1,1)$ cannot be set on *active*.
!T1,-1,ACTIV: the module in position $(1,-1)$ cannot be set on *active*.
!T2,1,ACTIV: the module in position $(2,1)$ cannot be set on *active*.
!T1,2,ACTIV: the module in position $(2,1)$ cannot be set on *active*.
!T2,-1,ACTIV: the module in position $(-1,2)$ cannot be set on *active*.
!T2,0,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T0,2,ACTIV: the module in position $(2,0)$ cannot be set on *active*.
!T2,2,ACTIV: the module in position $(2,2)$ cannot be set on *active*.
- **Postconditions**:
SINACT: the module changes its state to *inactive*.

Rule Name: Exchange rule for Bridges
Ponte su cui arriva un attivo Priority: 100
Preconditions: SBRIDG !W0,1,C001 C001 !E0,1 = C007 0000
Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000 C007 + 0000 0001

- **Rule**: Exchange rule for Bridges
- **Priority**: 100
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0,1)$ needs to be occupied.
!W0,1,C001 C001: the value of the counter C01 of the module has to be less than the value of the counter C01 of its north neighbor.
= C007 0000: the rule applies only if the value of the counter $C07$ of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its north neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its north neighbor into its own counter C12.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Exchange rule for Actives
Priority: 100
Preconditions: T0,-1,BRIDG V0,-1,C001 C001 = C007 0000
Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000
C007 + 0000 0001

- **Rule**: Exchange rule for Actives
- **Priority**: 100
- **Preconditions**:
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.
V0,-1,C001 C001: the value of the counter C01 of the module has to be bigger than the value of the counter C01 of its south neighbor.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- **Postconditions**:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Internal change of counters I
Priority: 100
Preconditions: SRINFO !(! E0,1 !T0,1,OBSTA) = C007 0001
Postconditions: C001 + C011 0000 C002 + C012 0000 SACTIV

- **Rule**: Internal change of counters I
- **Priority**: 100
- **Preconditions**:
SRINFO: it is applicable only to modules set in the *rinfo* state. !(!E0,1 !T0,1,OBSTA): the grid cell in position $(0, 1)$ needs to either empty or occupied by an obstacle.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- **Postconditions**:
SACTIV: the module changes its state to *active*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.

Rule Name: Internal change of counters II
Priority: 100
Preconditions: SRINFO !E0,1 !T0,1,OBSTA = C007 0001
Postconditions: C001 + C011 0000 C002 + C012 0000 SBRIDG

- **Rule**: Internal change of counters II
- **Priority**: 100
- **Preconditions**:
SRINFO: it is applicable only to modules set in the *rinfo* state.
!E0,1: the grid cell in position $(0, 1)$ needs to be occupied.

!T0,1,OBSTA: the grid cell in position $(0, 1)$ cannot be occupied by an obstacle.
= C007 0001: the rule applies only if the value of the counter C07 of the module is 1.
- Postconditions:
SBRIDG: the module changes its state to *bridge*.
C001 + C011 0000: the module copies the value of its counter C11 into its counter C01.
C002 + C012 0000: the module copies the value of its counter C12 into its counter C02.

```
Rule Name: Case of the last column (Bridge)
Priority: 100
Preconditions: SBRIDG T0,1,ACTIV = C001 C013 = C007 0000
!(!V0,1,C001 C013 W0,1,C001 C013)
Postconditions: SRINFO C011 + 0,1,C001 0000 C012 + 0,1,C002 0000
C007 + 0000 0001
```

- Rule: Case of the last column (Bridge)
- Priority: 100
- Preconditions:
SBRIDG: it is applicable only to modules set in the *bridge* state.
T0,1,ACTIV: the module in position $(0, 1)$ needs to be set on *active*.
!(!V0,1,C001 C013 W0,1,C001 C013): the value of the counter C01 of the module has to be different from the value of the counter C13 of its north neighbor.
= C001 C013: the rule applies only if the value of the counter C01 of the module is equal to the value of its counter C13.
= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- Postconditions:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C007 + 0000 0001: the modules sets to 1 its counter C07.

```
Rule Name: Case of the last column (Actives)
Priority: 100
Preconditions: SACTIV T0,-1,BRIDG < C001 C013 !V0,-1,C001 C013 W0,-1,C001 C013 = C007 0000
Postconditions: SRINFO C011 + 0,-1,C001 0000 C012 + 0,-1,C002 0000
C007 + 0000 0001
```

- Rule: Case of the last column (Actives)
- Priority: 100
- Preconditions:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,BRIDG: the module in position $(0, -1)$ needs to be set on *bridge*.
(!V0,-1,C001 C013 W0,-1,C001 C013): the value of the counter C01 of the module has to be equal to the value of the counter C13 of its south neighbor.
< C001 C013: the rule applies only if the value of the counter C01 of the module is less than the value of its counter C13.

= C007 0000: the rule applies only if the value of the counter C07 of the module is 0.
- Postconditions:
SRINFO: the module changes its state to *rinfo*.
C011 + 0,-1,C001 0000: the module stores the value of the counter C01 of its south neighbor into its own counter C11.
C012 + 0,-1,C002 0000: the module stores the value of the counter C02 of its south neighbor into its own counter C12.
C007 + 0000 0001: the modules sets to 1 its counter C07.

Rule Name: Counter C007
Priority: 15
Preconditions: SBRIDG E0,1 !(=C017 0000 =C007 0000)
Postconditions: C007 + 0000 0000 C017 + 0000 0000

- Rule: Counter C007
- Priority: 15
- Preconditions:
SBRIDG: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
< C001 C013: the rule applies only if the value of the counter C01 of the module is less than the value of its counter C13.
!(=C017 0000 =C007 0000): the rule does not apply to modules with the two counters C17 and C07 set to zero.
- Postconditions:
C017 + 0000 0000: the modules sets to 0 its counter C17.
C007 + 0000 0000: the modules sets to 0 its counter C07.

Rule Name: Counter C018
Priority: 150
Preconditions: SACTIV T0,-1,STOPP = C013 C001 !V0,-1,C001 0001 W0,-1,C001 0001 > C009 C010 = C019 0000
Postconditions: C018 + 0000 0001 C019 + 0000 0001

- Rule: Counter C018
- Priority: 150
- Preconditions:
SACTIV: it is applicable only to modules set in the *active* state.
T0,-1,STOPP: the grid cell in position $(0, -1)$ needs to be set in the *stop* state.
= C013 C001: the rule applies only if the value of the counter C01 meets the value of its counter C13.
> C009 C010: the rule applies only if the value of the counter C09 of the module is bigger than the value of its counter C10.
= C019 0000: the rule applies only if the value of the counter C19 is 0.
(!V0,-1,C001 0001 W0,-1,C001 0001): the value of the counter C01 of the module in position $(0, -1)$ has to be equal to 1.
- Postconditions:
C018 + 0000 0001: the modules sets to 1 its counter C18.
C019 + 0000 0001: the modules sets to 1 its counter C19.

Rule Name: Counting of the modules of the last column
Priority: 100
Preconditions: SBRIDG !E0,1 = C013 C001 = C016 0000 !V0,1,C018 0001 W0,1,C018 0001 > C009 C010
Postconditions: C014 + C014 0001 C016 + 0000 0001 C018 + 0000 0001

- **Rule**: Counting of the modules of the last column
- **Priority**: 100
- **Preconditions**:
SBRIDG: it is applicable only to modules set in the *bridge* state.
!E0,1: the grid cell in position $(0, 1)$ needs to be occupied.
= C013 C001: the rule applies only if the value of the counter C01 of the module equals the value of its counter C13.
= C016 0000: the rule applies only if the value of the counter C16 of the module is 0.
> C009 C010: the rule applies only if the value of the counter C09 of the module is bigger than the value of its counter C10.
(!V0,1,C018 0001 W0,1,C018 0001): the value of the counter C18 of the module in position $(0, 1)$ has to be equal 1.
- **Postconditions**:
C014 + C014 0001: the module adds 1 to its C04 counter.
C016 + 0000 0001: the module sets its C16 counter to 1.
C018 + 0000 0001: the modules sets to 1 its counter C18.

Rule Name: Auxiliar counter for the counting
Priority: 100
Preconditions: SBRIDG E0,1 = C016 0001
Postconditions: C016 + 0000 0000

- **Rule**: Auxiliary counter for the counting
- **Priority**: 100
- **Preconditions**:
SBRIDGE: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C016 0001: the rule applies only if the value of the counter C16 of the module is 1.
- **Postconditions**:
C016 + 0000 0000: the module adds 0 to its C04 counter.

Rule Name: Bridges reconfiguration
Priority: 100
Preconditions: SBRIDG E0,1 = C014 C015
Postconditions: SMOVES

- **Rule**: Bridges reconfiguration
- **Priority**: 100
- **Preconditions**:
SBRIDGE: it is applicable only to modules set in the *bridge* state.
E0,1: the grid cell in position $(0, 1)$ needs to be empty.
= C014 C015: the rule applies only if the value of the counter C14 of the module equals the value of its counter C15.
- **Postconditions**:
SMOVES: the module changes its state to *moves*.

Chapter 10

General obstacles

10.1 Goal

In this chapter we discuss the strategy for the overpassing of general obstacles, i.e. obstacles with general shapes formed by connected components which lay on the ground, without any height restriction. In the following, we will not focus neither on the initial shape of the system, nor in the reconfiguration at the end of the locomotion; the crossing can be performed by any system that is able to reconfigure into a worm shape as it touches the obstacle, or before, as long as the system starts its movement externally to the orthogonal convex hall of all obstacles.

10.2 Strategy

The strategy for the overpassing of a general connected obstacle is analogous to the one used in the locomotion of the histogram with obstacles: as the system encounters the obstacle, the modules create a static path over it, changing their state to *paths* and allowing the others to walk over and traverse it without any disconnection. In the case of general obstacles, though, many new difficulties are generated by the new possible configurations; as the obstacle is not configured as an histogram anymore, bottlenecks are created between different branches of the obstacle, and between different modules of the path. Notice that, as the obstacle is connected, the presence of a bottleneck is always associated to the presence of an hole or a cul-de-sac in the obstacle, which may cause complications. In order to overpass this type of structures and avoid disconnections and collisions, different types of bridges are created during the crossing, this time not only between the obstacle and the system, but between two modules of the system too. In the following sections, we analyze all the different types of bridges that are needed in order to deal with all the possible situations that the system may encounter during its way. We treat in the last sections the cases of combinations of different type of bridges, and the practical structure of the rules.

10.3 Bottlenecks of width 2

The first critical configuration for the formation of the path is the one generated by the presence of bottlenecks of width 2 created between two different branches of the obstacle. In order to overpass such bottlenecks, the first type of bridge has been introduced: we are for the first time in front of the need of the creation of bridges between modules of the path. As an active module detects the presence of a bottleneck of this type then, it changes its state to a bridge state and blocks the way to the rest of modules, avoiding the filling of the bottleneck and making the overpassing fluider and faster. Such bridge deals with all the cases depicted in Figure 10.1.

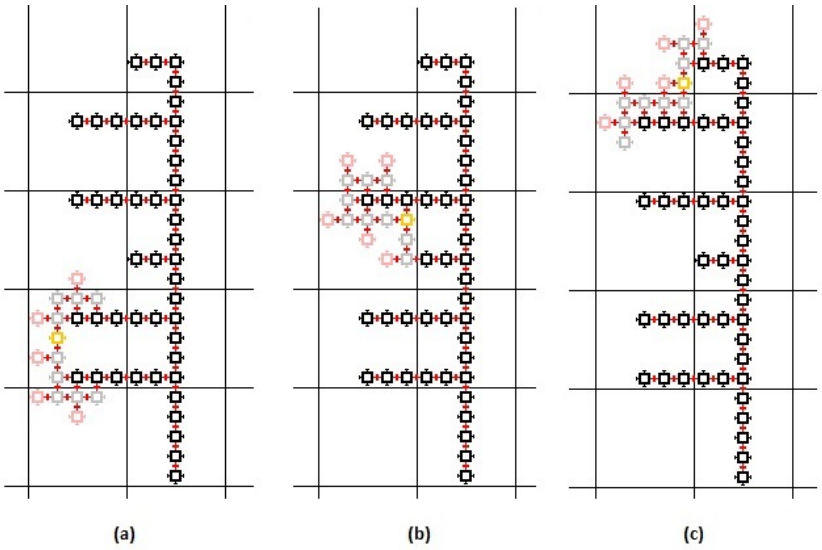


Figure 10.1: First type of bridge: as an active module encounters a bottleneck of width 2, it changes its state and stays still until all the rest of modules have crossed it; as it becomes the last module of the path, it activates another time and continues its locomotion. In the Picture we can see that the bridge is formed in all the three possible cases: in (a) the bridge is formed between two branches of the same length; in (b) the first branch that the system encounters is shorter than the second one, while in (c) the first one is longer than the second. The fact that in the Figure the two branches forming the bottlenecks are parallel is not a limitation, as the formation of the bridge is only related to the position of the two last module of the branches; the position of the rest of the obstacle modules is then irrelevant. Grey modules are modules of the path, the orange module is the bridge, and black modules are obstacle modules.

Notice that the same situations can be generated in all the different directions; such cases are controlled by three more analogous bridges, each one dedicated to a different possible orientation, and each one working exactly in the same way.

10.4 Bottlenecks of width 3: Introduction

The second important case to consider is the one in which a bottleneck of width 3 is created by two different modules of the obstacle. As the system creates the path over the obstacle, any bottleneck of width 3 is transformed in a bottleneck of width 1 formed between two different modules of the path, as we can notice in Figure 10.2.

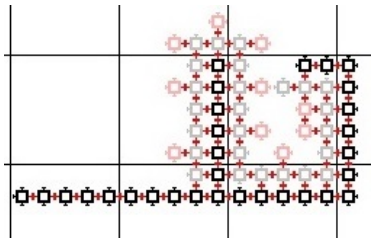


Figure 10.2: Example of bottleneck of width 3: as there is a 3-grid cells distance between the modules of the obstacle, the path formation generates a bottleneck of width 1, through which modules can walk in both directions colliding the one with the other. Grey modules are modules of the path, pink modules are active and black modules are obstacle modules.

The need of a creation of a bridge in this case is the need to control collisions between active modules that, while following the path, enter into the hole created by the obstacle, and modules that come out from it. The many particular situations that can be generated can be grouped into the four general cases depicted in Figure 10.3, depending on the different relative situations of the two modules of the path that create the bottleneck (i.e. on the different configurations of the obstacle that they are covering); notice that each of this case needs to be considered in all the possible orientations.

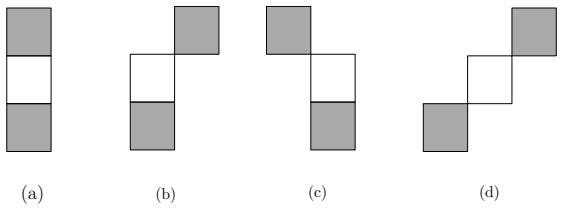


Figure 10.3: General cases of bottleneck of width 1 between modules of the path: such cases are generated by the formation of the path over bottlenecks of width three between two branches of the obstacle.

In the following sections we will treat separately the different cases, and present the different strategies created in order to overpass such type of configurations.

10.5 Bottlenecks of width 3: first case

In this section we treat the case (a) of Figure 10.3; in this configuration, as two branches of the obstacles form a bottleneck of width 3, the path formation generates a bottleneck of width 1 formed by two path modules placed in the same vertical (or horizontal, depending on the direction) line. Depending on the configuration of the system around the two path modules, different strategies are adopted.

10.5.1 Moving bridges

The first case to study is the one depicted in Figure 10.4; in this case, an active module is standing in the bottleneck formed by the two path modules, and a third path module blocks its way on one of its sides. The active module changes then its state to a bridge state.

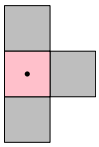


Figure 10.4: Case of the moving bridge: the active module changes its state to a bridge state when it finds itself stuck in a bottleneck of width 1 and with another path module blocking its way on its right.

As soon as the bridge is formed, the remaining active modules walk over it and continue the formation of the path; if the configuration of the new parts of the path is such that a new bottleneck of width 1 is formed next to the original one, the bridge slides to its left and blocks it; this is done until the end of one of the two branches is reached and the last bottleneck is blocked. We can see an example of such behavior in Figure 10.5.

As soon as all the modules have overpassed the bottleneck, the bridge activates again and clears the way for the modules in the hole to activate and move.

10.5.2 Joint bridges

The second type of bridge to analyze is the one created in the situations depicted in Figure 10.6; in this cases the obstacle creates an hole in which the active modules enter while creating the path; as the bottleneck is formed, some active modules are inside the hole and the rest of modules are outside, so the bottleneck needs to be blocked in order to avoid collisions.

As soon as the bridge is formed the remaining active modules walk over it and continue the formation of the path. Given the definition of path, when new parts of it are formed at least one more bottleneck of width 1 is created next to the original one; this is due to the presence of the path in relative position $(0, 1)$ with respect to the bridge (refer to Figure 10.6), and to the fact that $(1, 1)$ cannot be occupied by a path module: the module in $(0, 1)$ needs to be adjacent to an obstacle module, and such obstacle produces the formation of at least two

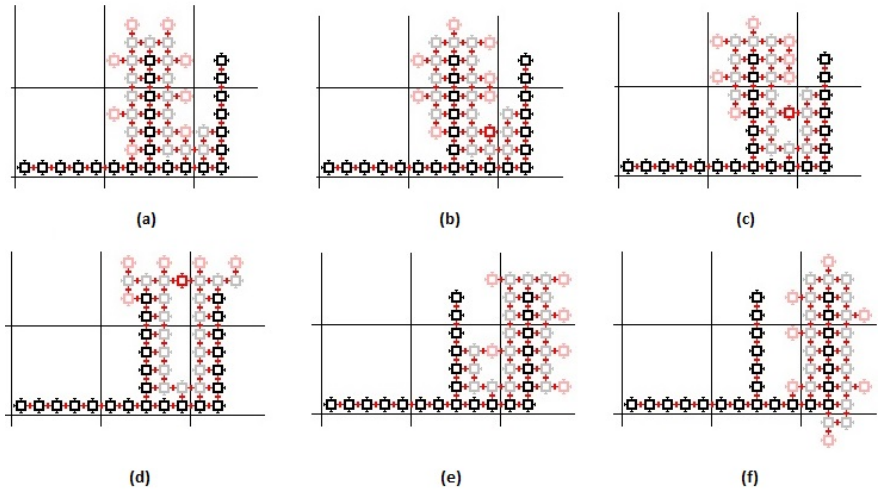


Figure 10.5: The moving bridge case: as the bridge is formed, if other bottlenecks are created next to the original one, the bridge moves under a specific rule and blocks all the bottlenecks until it reaches the end of one or both of the branches. The active modules that may cause collisions with the moving bridge stand still if the bridge is moving, due to a specific precondition. Grey modules are modules of the path, pink modules are active, black modules are obstacle modules and the red module is the moving bridge.

other adjacent path modules. When this type of bridge is formed then, other bridges of the same type join the first one; such additional bridges are created as long as more bottlenecks of width 1 are encountered; we can see and example of such configuration in Figure 10.7.

The main difference between this case and the case of moving bridges is that in this case there can be found active modules on both sides of the bridge: we don't allow the locomotion of the bridge, as its presence blocks the movement of the modules in the hole and avoid loops; as an active module in the hole encounters this type of bridge, in fact, it blocks its movement until the bridge activates again and clears the way.

We can see that with the formation of the moving bridge and the joint bridges we are able to deal with all the possible cases of a bottleneck of with 1 formed by two path modules positioned in the same vertical line, in the examined direction; by the application of the same strategies in the 3 directions left, we cover all the possible cases for the bottlenecks of width 1 formed by two vertically aligned path modules. In Figure 10.8 we can see a summary of all the possible cases for the current direction; it is easy to see that any other case is either a sub-case of the ones depicted, or the symmetric of one of them, or it is the same case in an horizontal version.

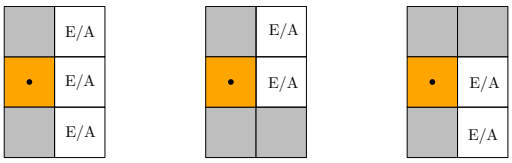


Figure 10.6: Second case of bridge for bottlenecks of width 3: obstacle modules, and the path over it, create an hole, and the first active module which encounters it blocks the way to the others in order to avoid collisions between the entering modules and the coming out modules. Grey modules are path modules, the orange cell is the bridge and E/A means that the grid cell is either empty or occupied by an active module. All the symmetric cases and their horizontal analogous are treated in the same way, through the formation of analogous bridges.

10.6 Bottlenecks of width 3: trasversal bridges

The second and the third cases of bottleneck (refer to (b) and (c) of Figure 10.3) are treated through the creation of a third type of bridge between the two modules of the path: the *trasversal bridge*. The different cases possible for this type of configuration are depicted in Figure 10.9; for each one of these cases, a different trasversal bridge is created. As soon as an active module encounters such configuration, it changes its state to a trasversal bridge state and stays still until the rest of module external to the hole have crossed the bottleneck; the active module in the internal part of the hole don't move until the bridge is active again and clears the way. The same four different cases of the Figure 10.9 can be found in their horizontal version, so eight different trasversal bridges need to be defined.

The process for the creation and reactivation of a trasversal bridge requires the joint action of an additional module, used as a pivot by the reactivating bridge in order to avoid disconnections; as we can see in Figure 10.10 (a), without an auxiliary module the bridge could not activate and move without generating a disconnection of the system. In this case, then, as an active module reaches the right position (on the top of the trasversal bridge in the case of Figure 10.10 (b)) it changes its state to *path* and conserves its position until the bridge is active again; it activates then, as soon as it is free to move.

The auxiliary path position depends on the position of the bridge with respect to the two path modules of the bottleneck; such auxiliary module is always formed outside the hole, by an active module that is crossing the bottleneck; in Figure 10.11 we can observe the different positions of such auxiliary module depending on the different configurations; notice that the formation of the bridge is performed always by active modules before they enter in the hole, and not by active modules that are coming out of it.

In Figure 10.12 we can observe the process of formation and reactivation of a trasversal bridge module.

Notice that a trasversal bridge is not formed in the cases in which the same bottleneck can be overpassed through the use of the bridges described in Sub-

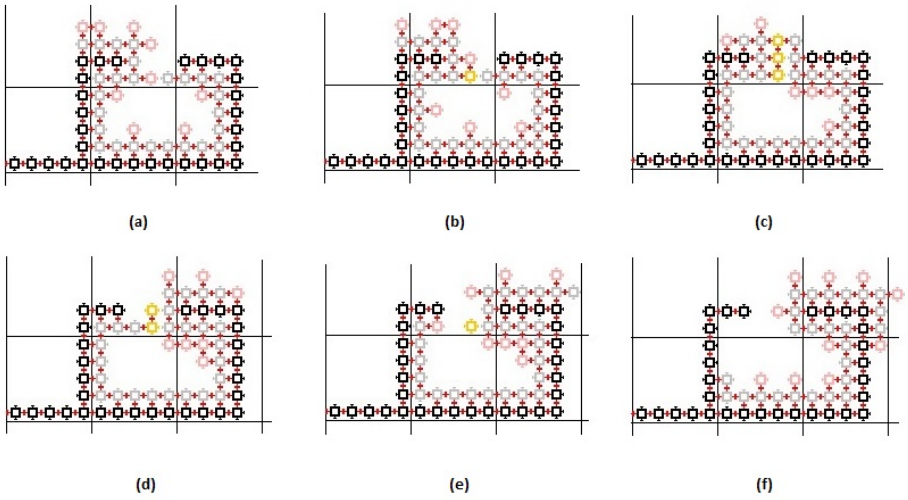


Figure 10.7: Joint bridges: as a bottleneck is formed in (a) the appropriate bridge is created in (b); in (c) we can see how two others analogous bridges blocks the new bottlenecks. In (d), (e), (f) the bridges reactivate and clear the way as all the modules out of the hole have crossed the bottleneck. Notice in (c), (d) and (e) how the active modules in the hole block their movement as the bridge is detected, and stay still until the bridge is again an active module. Case II and III are treated in the same way.

section 10.5.2. In the case of Figure 10.13 for example, and in all the analogous situations, although the module encounters a bottleneck of the type described for the trasversal modules, it continues its movement and forms a joint bridge in the following time step. In this case, a possible collision between the current module and any active module coming out from the hole is avoided through dedicated preconditions in the rules for the locomotion of active modules over the path. This choice is done in order to reduce the number of bridges, as the definition of new trasversal bridges would be needed in order to deal with these other configurations without the use of the previous bridges defined.

In Figure 10.14 we can see an example of a bottleneck of width 1 between two path modules crossed through the use of our method.

10.7 Diagonal bridges

The last case to study is the case of a bottleneck formed by two path modules diagonally aligned, as in Figure 10.3 (c).

In order to deal with this case, a new type of bridge is created: the *diagonal bridge*; such bridge is formed by the conjunction of two different modules that work together in order to block the way and avoid conflicts generated between modules coming out from the hole and modules that are entering in it. The creation of this type of bridges is divided in two phases; in the first phase of the formation of a diagonal bridge, an active module who recognizes the presence of

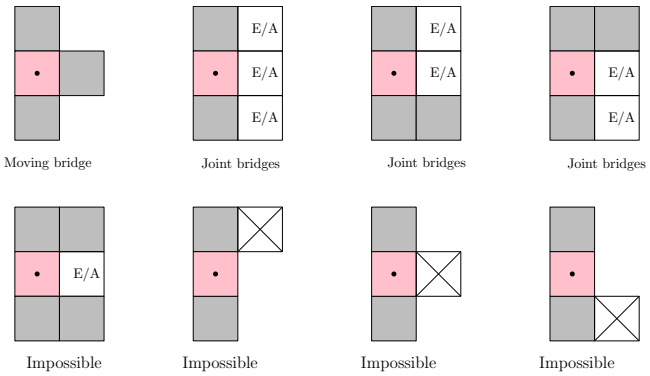


Figure 10.8: Possible cases for the bottlenecks of width 1 formed by two vertically aligned path modules. Grey modules are path modules, the active module candidate for the bridge formation is the pink module, E/A indicates that the grid cell can be either empty or occupied by an active module and crossed cells are obstacles. A grid cell without any specified condition can be empty or occupied by any possible module.

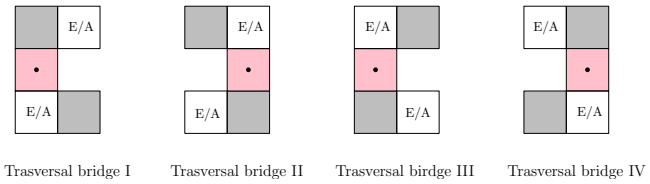


Figure 10.9: Transversal bridges: path modules are depicted in gray, the active module that encounters the bottleneck is depicted in pink; E/A means that the grid cell is either empty or occupied either by an active module.

the bottleneck stops and changes its state into a bridge state, creating the first component of the bridge. As soon as a second active module reaches the right position, and notices the presence of the first component of the bridge, it changes its own state creating the second one and completes the diagonal bridge. While the bridge is present, the active modules which are out of the hole overpass it by walking diagonally over these two modules, and continuing the formation of the path on the other side. As soon as all the modules have overpassed the bridge, its first component activates and enters the hole; the second component detects such activation and activates itself, clearing the way for the modules that are stuck into the hole. We can observe an example of the overpassing of a diagonal bottleneck with the use of diagonal bridges in Figure 10.15.

There are four different possible configurations that require the formation of a diagonal bridge, depending on the orientation of the path modules that form the bottleneck and from the relative position of the hole; we can see all these cases depicted in Figure 10.16, together with the position of the two component of the diagonal bridge formed in each case.

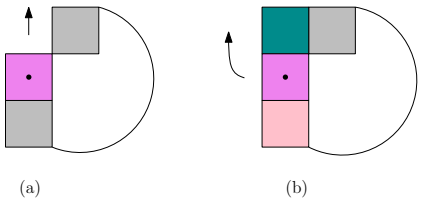


Figure 10.10: Formation of the auxiliary path module for trasversal bridges: (a) shows how, without the auxiliary path module, at the moment of activation the trasversal bridge (depicted in purple) cannot move as it cannot attach to any other module; in (b) we can see how, through the auxiliary path formation (blue module), the bridge is able to move and clear the way as soon as the first of the two path modules of the bottleneck (pink module) activates again.

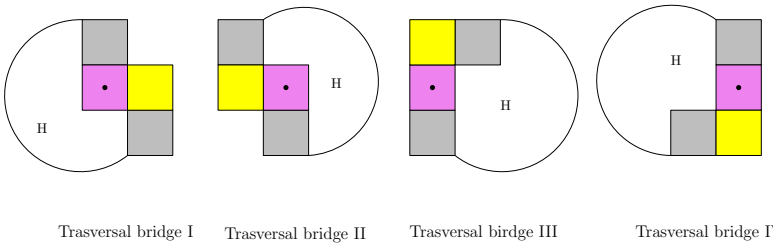


Figure 10.11: Different cases for the trasversal bridges in the vertical direction. Grey modules are path modules, pink modules are the different trasversal bridges, and yellow modules indicate the auxiliary path modules. The curved segment that links the two path modules in each configuration indicates the position of the hole.

10.8 Combinations of bridges

In the previous section we discussed the strategy for the formation of bridges without considering the possibility of the presence of two or more different types of bridges combined together in the same configuration. The most versatile type of bridge in this sense is the first bridge discussed in Section 10.3: such bridge is treated exactly as a path module, and is then recognized by other modules as a possible support module for the creation of a bridge: any module can block a bottleneck formed between two path module as between a path module and a bridge of type I. We can observe many configurations in which two consecutive bridges of the first type are created, or in which bridges of other kinds are created over a bridge of the first type; as an example of this property see Figure 10.17.

This first kind of bridge is then such that any other module is able to attach to it an form any bridge; this is the only kind of bridge which has this property in our settings. The rest of combinations are allowed only between selected pairs of bridges in order to deal with particular situations, as in Figure 10.18 where a bridge of type II is formed between a path module and a diagonal bridge.

The rest of combinations allowed in our settings are all combinations in the sense that two different kinds of bridges can be created in the same neighbor-

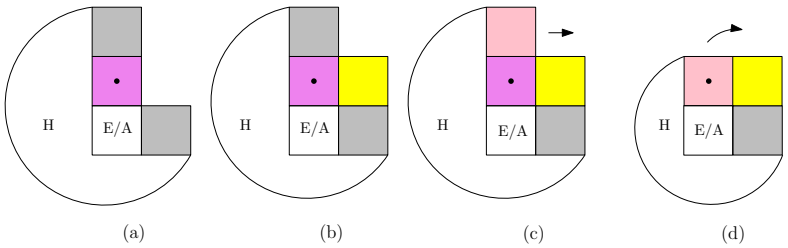


Figure 10.12: The process of reactivation of a trasversal bridge through its auxiliary module: in (a) the bridge is formed as soon as the bottleneck is detected; in (b) the first active module changes its state to *path* forming the auxiliary module for a bridge. In (c) and (d), as the first one of the modules of the bottleneck is active again and clears the way, the bridge activates and uses the auxiliary module as a pivot to move again. The bridge is the violet module, path modules are gray and active modules are pink; we depict the auxiliary module in yellow in order to distinguish it from the rest of path modules.

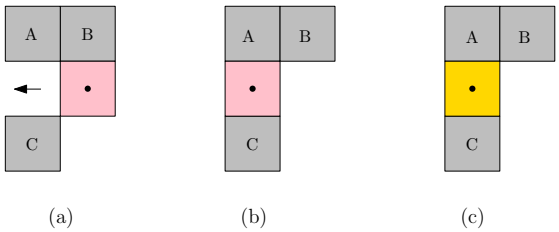


Figure 10.13: If the same bottleneck can be overpassed by the formation of a joint bridge, the current module continue its movement. In (a) and (b) we can see how, although the module detects a trasversal bottleneck formed by modules B and C, due to the presence of the path module A it continues its movement and chooses to form another type of bridge. Path modules are gray, the current active module is pink and the bridge is depicted in orange.

hood, but they cannot be attached the one to the other. The only requirement for the formation of the rest of bridges is than that they need to be able to recognize the presence of other ones in their neighborhood, and to move in this case as in the presence of path modules.

10.9 Strategy for the implementation of the rules

In order to deal with the presence of four different directions of movement, some particular features are introduced to the structure of the rules; we present here the general idea of the rules created for the overpassing of general connected obstacles. The first thing to notice is that in this settings each definition of a bridge corresponds to a new bridge state; such states are named following different patterns depending on the type of bridge defined: the pattern **brid* is used for the moving bridges, *brid** for the joint bridges, *bris** for the bridges

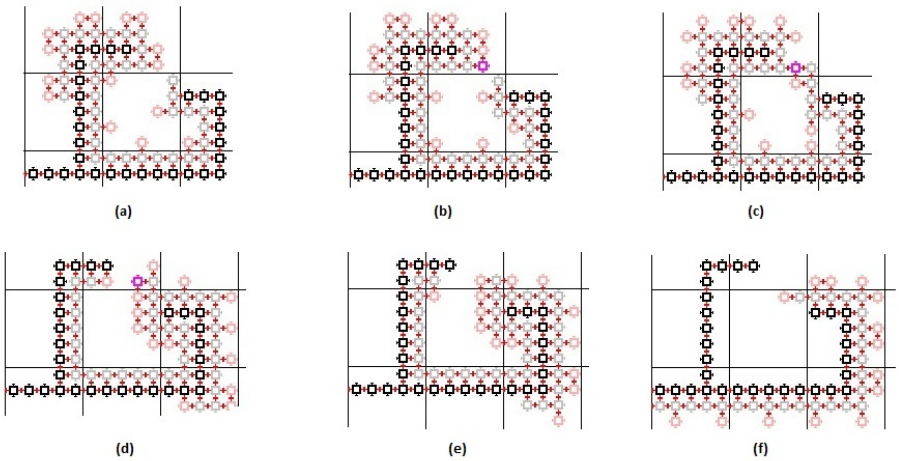


Figure 10.14: Example of crossing of a bottleneck through the use of a trasversal bridge: as an active module encounters the bottleneck in (a), it changes its state to a bridge state in (b). In (c) an auxiliary active module changes its state to path in order to allow the reactivation and movement of the bridge. In (d) and (e) the bridge is free to move and activates again; in (f) we can see how the crossing is completed.

for bottlenecks of width 2, *tras** for the trasversal bridges and *diag** for the diagonal ones; the first or last missing letter, indicated with *, is replaced by a different letter for each different orientation and case. The definition of each bridge is always associated with the definition of some advance rules specific for the movement over such bridge, that cannot be applied over any other; this is due to the fact that now active modules move in the four different directions, and we control the directions that modules have to follow through the definition of different advance rules for different bridges. To understand this, we can consider as an example of this strategy the formation of the bridge *bridf* depicted in Figure 10.7; notice how the active modules out of the hole continue their movement, while the active modules inside the hole stop and stand in line, waiting for the bridge to be active again; this behavior is due to the fact that, while there exists a 'North rule over the bridf', no 'South rule over the bridf' is defined, so any module which intends to move south cannot apply any rule if the support module is set to *bridf*, and has to wait until the bridge disappears. In order to make this system work, it is really important that each bridge is formed in the right place, and that a bridge designed for a given direction is not created for another one; this, in some cases, is difficult or impossible to achieve through the only knowledge of the local configuration of the path. We can see an example of this ambiguity comparing the two cases depicted in Figure 10.19, in which the preconditions for the formation of the two bridges *bridf* and *bridh* are shown, together with the relative positions of the holes. In this case, the preconditions for the formation of two different bridges are not incompatible, so any active module which encounters such configuration is not able to understand

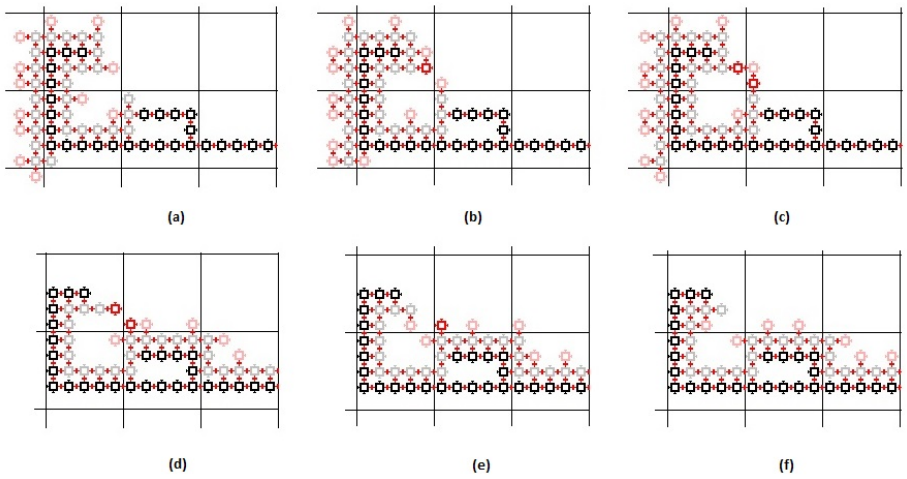


Figure 10.15: Example of the overpassing of a hole through the formation of a diagonal bridge. In (a), the first active modules to detect the bottleneck is the one which forms in (b) the first component of the bridge. In (c) the second component of the bridge is created. In (d) and (e) we can observe the reactivation of the two components.

which of the state is the right one, as active modules cannot deduce the position of the hole just from the position of its neighbors.

To deal with such kind of problems, a new tool is introduced for the control over the identity of the last path module: each time a module changes its state to path, its counter C00 is set to 1, and then reset to zero as the following path module is created; at each step of the locomotion of the system then, modules have a way to check which is the last module of the path, and in ambiguous situations a check over the counter C00 of the right path module of the bottleneck solves the problem of the orientation. In the cases of Figure 10.19, for example, the *bridf* bridge will be created if the path module in relative position $(0, 1)$ has its C00 set to 1, while the *bridh* will be chosen if the last module of the path is the one in relative position $(0, -1)$. The change of the counter C00 is applied to the case of bridges for the bottlenecks of width 2: as they are physically part of the path, in spite of their state they are treated as modules of the path in the strategy of the movement.

10.10 Still to be done

In the next section we present an incomplete version of the rules for the overpassing of general connected obstacles. Although the different kind of bottlenecks presented in the previous sections cover all the possible cases, and in spite of the fact that the defined correspondent bridges are able to deal with each one of these bottlenecks one by one, when such bottlenecks and bridges combine together in the same neighborhood, the path over them can generate many dif-

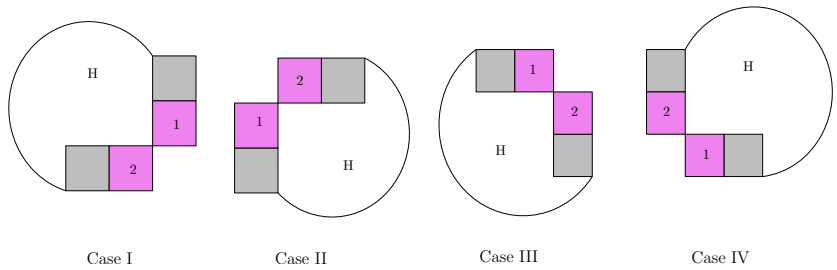


Figure 10.16: Different cases for the diagonal bridge formation, with the chronological order for the formation of the components. Grey modules are path modules, pink modules are the components of the bridge; the H and the line that connects the path modules of each case is the indication of the relative position of the hole.

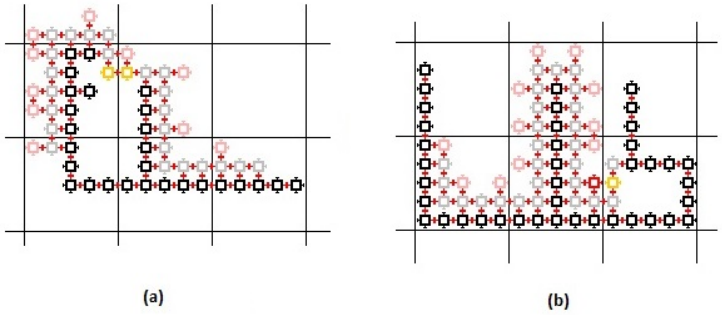


Figure 10.17: Two examples of combinations for bridges of the first type: in (a) we can observe two consecutive bridges of type I created along the path, while in (b) a bridge of type II (red in the Picture) is formed over a bridge of type I. Orange modules are bridges of type I, red modules are bridge of type II, gray modules are path modules and black modules are obstacles.

ferent configurations, due to the complexity of the shapes that the obstacle may take. Such diverse cases have not been treated yet in a systematic way, and the rules reflect this lack of orderliness. In order to deal with all the different configurations then, more different rules of activation of the path and bridges may be necessary, and the introduction of new different combinations of bridges may be needed. As the set of rules presented is not a definitive version, we skip the studies of correctness and complexity and present the current rules without a detailed explanation of each one of them.

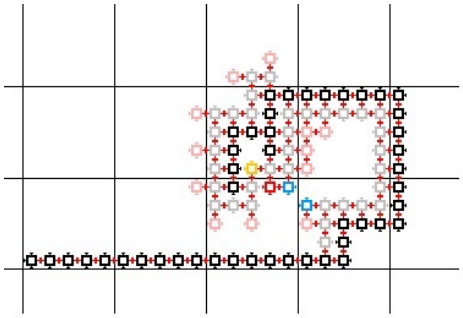


Figure 10.18: Combination between a diagonal bridge and a bridge of type II; the diagonal bridge is depicted in blue, while the red module is a bridge of type II. As always, gray modules are path modules and black modules are obstacles.

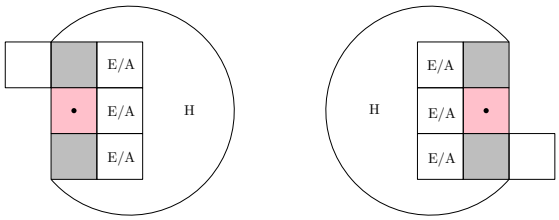


Figure 10.19: Preconditions for the formation of two different bridges: the two situations are not incompatible, and any module which satisfy the one set of preconditions could at the same time satisfy the second. We indicate the position of the hole too, in order to point out how the confusion between the two bridges could affect the efficacy of our strategy of movement. Notice that in the preconditions we are not depicting the cases of combinations of bridges.

10.11 Rules

Nord
2
!(!SACTIV !SINACT) N001* T1,1,INACT
P0,1 SACTIV A****

Nordest
2
!(!SACTIV !SINACT) N001* T1,0,INACT E1,1
P1,1 SACTIV A***1

Est
2
!(!SACTIV !SINACT)N0*01 T1,-1,INACT
P1,0 SACTIV A***1

Sud
2
!(!SACTIV !SINACT) N*1*0 T-1,-1,INACT !(!T1,0,OBSTA !E1,0)
P0,-1 SACTIV A***1

Sudest
2
!(!SACTIV !SINACT) N0001 T0,-1,INACT E1,-1
P1,-1 SACTIV A*1**

Disattivazione
1
SACTIV N*10* !T0,1,DIAG* !T-1,0,DIAG* !T0,-1,PATHS !T0,1,PATHS !T-1,0,PATHS !T1,0,PATHS !T0,1,*BRID !T0,-1,*BRID !T1,0,*BRID !T-1,0,*BRID !T0,-1,ACTIV !T0,1,BRID* !T0,-1,BRID* !T1,0,BRID* !T-1,0,BRID* !T0,1,BRIS* !T0,-1,BRIS* !T1,0,BRIS* !T-1,0,BRIS*
SINACT

Counter for the last module of the path I
100
!(!SPATHS !SBRIS*) = C000 0001 !(!(T-1,0,PATHS T1,0,PATHS) !(T0,1,PATHS T0,-1,PATHS))
C000 + 0000 0000

Counter for the last module of the path II
100
!(!SPATHS !SBRIS*) = C000 0001 !(!(T-1,0,PATHS T0,-1,PATHS) !(T-1,0,PATHS T0,1,PATHS) !(T0,1,PATHS T1,0,PATHS) !(T1,0,PATHS T0,-1,PATHS))
C000 + 0000 0000

Counter for the last module of the path III
100

!(!SPATHS !SBRIS*) = C000 0001 !(!T0,1,PATHS !T0,1,BRIS*) T1,0,OBSTA
E1,-1
C000 + 0000 0000

Counter for the last module of the path - Case of consecutive bridges I
100
!(!SPATHS !SBRIS*) = C000 0001 !(!(T-1,0,BRIS* T1,0,PATHS) !(T-1,0,PATHS
T1,0,BRIS*) !(T0,1,BRIS* T0,-1,PATHS) !(T0,1,PATHS T0,-1,BRIS*))
C000 + 0000 0000

Counter for the last module of the path - Case of consecutive bridges II
100
!(!SPATHS !SBRIS*) = C000 0001 !(!(T-1,0,BRIS* T0,-1,PATHS) !(T-1,0,PATHS
T0,-1,BRIS*) !(T-1,0,BRIS* T0,1,PATHS) !(T-1,0,PATHS T0,1,BRIS*) !(T0,1,BRIS*
T1,0,PATHS) !(T0,1,PATHS T1,0,BRIS*) !(T1,0,BRIS* T0,-1,PATHS) !(T1,0,PATHS
T0,-1,BRIS*))
C000 + 0000 0000

First module of the path Case I
5
SACTIV N*110 T1,0,OBSTA T-1,0,INACT E-1,-1
SPATHS C000 + 0000 0001

First module of the path Case II
5
SACTIV N*111 T0,-1,INACT T1,0,OBSTA T-1,0,INACT
SPATHS C000 + 0000 0001

First module of the path Case III
5
SACTIV N**11 T1,0,OBSTA !(!E-1,0 !T-1,0,ACTIV) T0,-1,INACT
SPATHS C000 + 0000 0001

Path formation I
5
SACTIV T1,0,PATHS T0,1,OBSTA
SPATHS C000 + 0000 0001

Path formation II
5
SACTIV T0,-1,PATHS T1,0,OBSTA
SPATHS C000 + 0000 0001

Path formation III
5
SACTIV T0,-1,PATHS T1,0,OBSTA
SPATHS C000 + 0000 0001

Path formation IV
5
SACTIV T-1,0,PATHS T0,-1,OBSTA
SPATHS C000 + 0000 0001

Path formation V
5
SACTIV T0,1,PATHS T-1,0,OBSTA
SPATHS C000 + 0000 0001

East Angle I
5
SACTIV T1,0,PATHS E0,1 T1,1,OBSTA
SPATHS C000 + 0000 0001

East Angle II
5
SACTIV T0,-1,PATHS E1,0 T1,-1,OBSTA
SPATHS C000 + 0000 0001

North Angle I
5
SACTIV T0,1,PATHS E0,-1 T-1,1,OBSTA
SPATHS C000 + 0000 0001

North Angle II
5
SACTIV T-1,0,PATHS E0,1 T1,1,OBSTA
SPATHS C000 + 0000 0001

West Angle I
5
SACTIV T-1,0,PATHS E0,-1 T-1,-1,OBSTA
SPATHS C000 + 0000 0001

West Angle II
5
SACTIV T1,0,PATHS E-1,0 T-1,1,OBSTA
SPATHS C000 + 0000 0001

South Angle I
5

SACTIV T0,-1,PATHS E1,0 T1,-1,OBSTA
SPATHS C000 + 0000 0001

South Angle II
5
SACTIV T-1,0,PATHS E1,0 T-1,-1,OBSTA
SPATHS C000 + 0000 0001

North on the Path
6
!(!SINACT !SACTIV) N0*1* !(!T1,0,PATHS !T1,0,INACT) T1,1,PATHS !(T0,2,BBRID T-1,1,PATHS) !(T0,2,BBRID !E-1,1) !(T0,2,ACTIV !(!T-1,1,ACTIV !T-1,1,PATHS) T-1,0,PATHS E-1,-1) !(!(SACTIV T-1,0,ACTIV) !T-1,0,OBSTA !E-1,0 !(T-1,0,PATHS E-1,1 T-1,-1,PATHS)) !(T-1,1,ABRID E-1,0) !T0,2,ABRID
P0,1 A111* SACTIV

North-East on the Path
7
SACTIV N0*1* T1,0,PATHS E1,1 !(T2,1,ABRID !(!T1,2,PATHS !T1,2,ACTIV)) !(T1,2,ACTIV T0,2,PATHS)
P1,1 A1*11

East
7
SACTIV N**01 T1,-1,PATHS !(T2,0,ACTIV T2,1,DIAG*) !(T2,0,ABRID !(!T1,1,PATHS !T1,1,ACTIV)) !T0,-1,ACTIV !T0,-1,BRID* !T0,-1,BRIS* !T0,-1,TRAS* !(T0,1,PATHS T0,-1,PATHS T1,1,PATHS T1,-1,PATHS E-1,-1) !(T1,1,PATHS T2,1,ACTIV) !T2,0,DBRID
P1,0 A*011

South
6
SACTIV N*1*0 T-1,-1,PATHS T-1,0,PATHS !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV)) !(T0,-2,ACTIV T1,-2,PATHS) !(T1,0,PATHS !SACTIV T0,1,PATHS T-1,0,PATHS) !(E0,1 E-1,1 T1,1,PATHS T-1,-1,PATHS) !(T1,-2,ACTIV T1,-1,PATHS) !T0,-2,CBRID
P0,-1 A*1*1

South-East
6
SACTIV N**01 E1,-1 T0,-1,PATHS !(T1,-2,DBRID !(!T2,-1,PATHS !T2,-1,ACTIV)) !(T0,-1,PATHS T1,-2,ACTIV) !(T2,0,ACTIV T2,1,PATHS E1,1) !(T1,-2,PATHS !E2,-1 !(!T0,-2,TRAS* !E0,-2 !T0,-2,ACTIV))
P1,-1 A*1*1

West
6
SACTIV N10** T-1,1,PATHS !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV)) !T0,1,TRAS* !(T-1,-1,PATHS T-2,0,ACTIV) !T-2,0,BBRID

P-1,0 A11*1

North-west
6
SACTIV N10** E-1,1 T0,1,PATHS !(T-1,2,BBRID !(!T-2,1,PATHS !T-2,1,ACTIV))
!(T-2,1,PATHS T-2,2,ACTIV)
P-1,1 A1*1*

North-west for Inactives
6
SINACT N10*0 E-1,1 !(!T1,0,OBSTA !E1,0)
SACTIV P-1,1 A1*1*

East for inactives
6
SINACT N0001 T1,-1,PATHS
SACTIV P1,0 A**11

North-east for inactives
6
SINACT N0011 T1,0,PATHS E1,1
SACTIV P1,1 A**11

South-west
6
SACTIV N*1*0 !(!E0,1 !T0,1,PATHS !(T0,1,ACTIV T-1,1,PATHS)) T-1,0,PATHS
E-1,-1 !(T-2,-1,CBRID !(!T-1,-2,PATHS !T-1,-2,ACTIV)) !(T0,-2,ACTIV T1,-
1,PATHS)
P-1,-1 A11**

South-west II
16
SACTIV N11*0 T-1,0,PATHS T0,1,OBSTA !(!T1,0,OBSTA !E1,0) E-1,-1
P-1,-1 A11*1

Path reactivation I
2
SPATHS N101* T0,1,PATHS T1,0,OBSTA !(!T0,-1,OBSTA !E0,-1)
SACTIV

Path reactivation I
2
SPATHS N0011 T1,0,PATHS T0,-1,OBSTA
SACTIV

Path reactivation III
2
SPATHS N11*0 T-1,0,PATHS T0,1,OBSTA !(!T1,0,OBSTA !E1,0)
SACTIV

Path reactivation IV
2
SPATHS N100* T0,1,PATHS T1,1,OBSTA !(!T0,-1,OBSTA !E0,-1)
SACTIV

Path reactivation V
2
SPATHS N0*10 T1,0,PATHS T1,-1,OBSTA !(!T-1,0,OBSTA !E-1,0)
SACTIV

Path reactivation VI
2
SPATHS N*101 T0,-1,PATHS T-1,0,OBSTA !(!T0,1,OBSTA !E0,1)
SACTIV

Path reactivation VII
2
SPATHS N01*0 T-1,0,PATHS !(!T1,0,OBSTA !E1,0) T-1,1,OBSTA
SACTIV

Path reactivation VIII
2
SPATHS N0*11 T1,0,PATHS !(!T-1,0,OBSTA !E-1,0) T0,-1,OBSTA
SACTIV

Path reactivation IX
2
SPATHS N0001 T0,-1,PATHS T-1,-1,OBSTA
SACTIV

Path reactivation X
2
SPATHS N1001 T0,1,OBSTA T0,-1,PATHS E1,-1
SACTIV

Bridge E Formation
10
SACTIV !(!T1,1,OBSTA !T0,1,OBSTA !T-1,1,OBSTA) E1,0 !(!E1,-1 !T1,-1,OBSTA) !T-1,0,OBSTA !(!(!E0,-1 !T0,-1,OBSTA) !T-1,0,PATHS !T-1,0,BRIS*) !(!T0,-1,OBSTA !T0,-1,PATHS !T0,-1,BRIS*)

SBRISE C000 + 0000 0001

Reactivation BridE I
5
SBRISE N10** T0,1,PATHS E1,0 !(!E0,-1 !T0,-1,OBSTA) E-1,1
SACTIV P-1,1 A**1*

Reactivation BridE II
5
SBRISE N*1*0 T-1,0,PATHS !(!E1,0 !T1,0,OBSTA) E-1,-1
SACTIV P-1,-1 A**1*

Reactivation BridE III
5
SBRISE N10*0 T0,1,PATHS !(!T-1,1,PATHS !T-1,1,*BRID !T-1,1,BRIS* !T-1,1,BRID*) !(!E1,0 !T1,0,OBSTA) E-1,0
SACTIV P-1,0 A**1*

Path over a BridE
5
SACTIV T0,-1,BRISE T1,0,OBSTA
SPATHS C000 + 0000 0001

Path over a BridE II
5
SACTIV T1,0,BRISE !(!T1,1,OBSTA !T0,1,OBSTA)
SPATHS C000 + 0000 0001

North over a BridE I
2
!(!SACTIV !SINACT) N0*1* T1,0,PATHS T1,1,BRISE !T-1,0,INACT !(SINACT T-1,0,ACTIV) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
P0,1 SACTIV A****

North over a BridE II
2
!(!SACTIV !SINACT) N0*1* T1,0,BRISE !T-1,0,INACT T1,1,PATHS !(SINACT T-1,0,ACTIV) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
P0,1 SACTIV A****

North-east over a BridE
2
!(!SACTIV !SINACT) N0*1* T1,0,BRISE E1,1 !(!E-1,0 !(T-1,0,PATHS SACTIV)) !(SINACT T-1,0,ACTIV)
P1,1 SACTIV A***1

East over a BridE
2
SACTIV N0*01 T1,-1,BRISE
P1,0 SACTIV A***1

North-west reactivation over a BridE (Path reactivation and movement)
6
SPATHS N10** E-1,1 T0,1,BRISE !(!T1,0,OBSTA !E1,0) !(!T0,-1,OBSTA !E0,-1)
P-1,1 SACTIV A1*1*

West reactivation over a BridE (Path reactivation and movement)
2
SPATHS N10** !(!E0,-1 !T0,-1,OBSTA) T0,1,BRISE T-1,1,PATHS !(!T1,0,OBSTA !E1,0) !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 SACTIV A11*1

North reactivation over bridE (Path reactivation and movement)
10
SPATHS N001* !T0,-1,PATHS T1,0,BRISE T1,1,PATHS !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
SACTIV P0,1 A****

BridN Formation
10
SACTIV !(!T-1,1,OBSTA !T-1,0,OBSTA !T-1,-1,OBSTA) E0,1 !(!E1,1 !T1,1,OBSTA) !T0,-1,OBSTA !(!(!E1,0 !T1,0,OBSTA) !T0,-1,PATHS !T0,-1,BRID* !T0,-1,BRIS*) !(!T1,0,OBSTA !T1,0,PATHS !T1,0,BRID* !T1,0,BRIS*)
SBRISN C000 + 0000 0001

Reactivation BridN I
5
SBRISN !(!E1,0 !T1,0,OBSTA) E0,-1 E-1,-1
P-1,-1 A1*** SACTIV

Reactivation BridN II
5
SBRISN E1,0 !(!T0,-1,PATHS !T0,-1,BRIS* !T0,-1,BRID*) E1,-1
P1,-1 A1*** SACTIV

Reactivation BridN III
5

SBRISN E0,-1 T-1,0,PATHS !(!T-1,-1,PATHS !T-1,-1,*BRID !T-1,-1,BRID* !T-1,-1,BRIS*) !(!E1,0 !T1,0,OBSTA) !(!E0,1 !T0,1,OBSTA)
P0,-1 A*1** SACTIV

Path formation over a BridN I
5
SACTIV T1,0,BRISN T0,1,OBSTA
SPATHS C000 + 0000 0001

Path formation over a BridN II
5
SACTIV !(!T-1,0,OBSTA !T-1,1,OBSTA) T0,1,BRISN
SPATHS C000 + 0000 0001

North over a BridN
2
!(!SACTIV !SINACT) N001* T1,1,BRISN !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
P0,1 SACTIV A****

West over a BridN I
2
SACTIV N10*0 T-1,1,BRISN !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 A11*1

North-west over a BridN
2
SACTIV N10** E-1,1 T0,1,BRISN
P-1,1 A1*1*

West over a BridN (Path reactivation and movement)
2
SPATHS N10*0 T0,1,BRISN !T1,0,PATHS !T1,0,BRIS* !T1,0,BRID* T-1,1,PATHS !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 A11*1 SACTIV

South-west over a BridN (Path reactivation and movement)
6
SPATHS N*1*0 !(!E0,1 !T0,1,OBSTA) !(!E1,0 !T1,0,OBSTA) T-1,0,BRISN E-1,-1
P-1,-1 A11** SACTIV

South over a BridN (Path reactivation and movement)
6
SPATHS N*1*0 !(!T-1,-1,PATHS !T-1,-1,BRID* !T-1,-1,BRIS*) T-1,0,BRISN !(!E0,1 !T0,1,OBSTA) !(!E1,0 !T1,0,OBSTA) !(T0,-2,DBRID !(!T1,-1,PATHS

!T1,-1,ACTIV))
P0,-1 A*1*1 SACTIV

BridO Formation
10
SACTIV !(!T-1,-1,OBSTA !T0,-1,OBSTA !T1,-1,OBSTA) E-1,0 !(!E-1,1 !T-1,1,OBSTA) !T1,0,OBSTA !(!(!E0,1 !T0,1,OBSTA) !T1,0,PATHS !T1,0,BRID* !T1,0,BRIS*) !(!T0,1,PATHS !T0,1,BRID* !T0,1,BRIS* !T0,1,OBSTA)
SBRISO C000 + 0000 0001

Reactivation BridO I
5
SBRISO !(!E0,1 !T0,1,OBSTA) E1,0 E1,-1
SACTIV P1,-1 A*111

Reactivation BridO II
5
SBRISO E0,1 T1,0,PATHS E1,1
SACTIV P1,1 A**11

Reactivation BridO III
5
SBRISO !(!E0,1 !T0,1,OBSTA) !(!T-1,0,OBSTA !E-1,0) !(!T0,-1,PATHS !T0,-1,BRISS) !(!T1,-1,PATHS !T1,-1,*BRID !T1,-1,BRID* !T1,-1,BRIS*) E1,0
SACTIV P1,0 A**11

Path formation over a BridO
5
SACTIV T0,1,BRISO T-1,0,OBSTA
SPATHS C000 + 0000 0001

Path formation over a BridO II
5
SACTIV T-1,0,BRISO !(!T0,-1,OBSTA !T-1,-1,OBSTA)
SPATHS C000 + 0000 0001

South over a BridO
6
SACTIV N*1*0 T-1,0,PATHS T-1,-1,BRISO !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV))
P0,-1 A*1*1

South over a BridO II
6
SACTIV N*1*0 T-1,0,BRISO T-1,-1,PATHS !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV))

P0,-1 A*1*1

South-west over a BridO
6
SACTIV N*1*0 T-1,0,BRISO E-1,-1
P-1,-1 A11**

West over a BridO
6
SACTIV N10** T-1,1,BRISO !(T-2,0,BBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 A11*1

South-east over a BridO
6
SACTIV N**01 T0,-1,BRISO E1,-1 !(!E0,1 !T0,1,OBSTA)
P1,-1 A*111

South-east reactivation over a BridO (Path reactivation and movement)
6
SPATHS N**01 T0,-1,BRISO !(!E0,1 !T0,1,OBSTA) !(!E-1,0 !T-1,0,OBSTA)
E1,-1
SACTIV P1,-1

East reactivation over a BridO (Path reactivation and movement)
2
SPATHS N0*01 T0,-1,BRISO T1,-1,PATHS !(!T-1,0,OBSTA !E-1,0) !(T2,0,ABRID
!(!T1,1,PATHS !T1,1,ACTIV))
P1,0 SACTIV A1*11

South reactivation over BridO (Path reactivation and movement)
10
SPATHS N*100 !T0,1,PATHS T-1,0,BRISO T-1,-1,PATHS !(T0,-2,DBRID !(!T1,-
1,PATHS !T1,-1,ACTIV))
SACTIV P0,-1 A*1*1

BridS Formation
10
SACTIV !(!T1,-1,OBSTA !T1,0,OBSTA !T1,1,OBSTA) E0,-1 !(!E-1,-1 !T-1,-
1,OBSTA) !T0,1,OBSTA !(!(!E-1,0 !T-1,0,OBSTA) !T0,1,PATHS !T0,1,BRID*
!T0,1,BRIS*) !(!T-1,0,OBSTA !T-1,0,PATHS !T-1,0,BRID* !T-1,0,BRIS*)
SBRISS C000 + 0000 0001

Reactivation BridS I
5
SBRISS N0*1* !(!T-1,0,OBSTA !E-1,0) E0,1 E1,1
SACTIV P1,1 A1*11

Reactivation BridS II
5
SBRISS E-1,0 !(!T0,1,PATHS !T0,1,BRID* !T0,1,BRIS*) E-1,1
SACTIV P-1,1 A**11

Reactivation BridS III
5
SBRISS E0,1 !(!E-1,0 !T-1,0,OBSTA) !(!T0,-1,OBSTA !E0,-1) T1,0,PATHS !(!T1,1,PATHS !T1,1,*BRID !T1,1,BRID* !T1,1,BRIS*) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
SACTIV P0,1 A**1*

Path formation over a BridS I
5
SACTIV T-1,0,BRISS T0,-1,OBSTA
SPATHS C000 + 0000 0001

Path formation over a BridS II
5
SACTIV T0,-1,BRISS !(!T1,0,OBSTA !T1,-1,OBSTA)
SPATHS C000 + 0000 0001

North over Brids
6
SACTIV N0*1* T1,0,BRISS !(!T-1,0,OBSTA !E-1,0) T1,1,PATHS !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
P0,1 A**1*

Nordest Brids
2
SACTIV N0*1* T1,0,BRISS E1,1
P1,1 SACTIV A***1

East over a BridS
6
SACTIV N**01 !(!T1,-1,BRISS !T0,-1,BRISS) !(!T1,-1,BRISS !T1,-1,PATHS) !(T2,0,ABRID !(!T1,1,PATHS !T1,1,ACTIV))
P1,0 A*011

South-east over a BridS
6
SACTIV N**01 T0,-1,BRISS E1,-1 !(!E0,1 !T0,1,OBSTA)
P1,-1 A*111

South over a BridS

6
SACTIV N*1*0 T-1,-1,BRISS !(!E1,0 !T1,0,OBSTA) !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV))
P0,-1 A*1*1

East over a BridS (Path reactivation and movement)
2
SPATHS N0*01 T0,-1,BRISS !T-1,0,PATHS !T-1,0,BRID* T1,-1,PATHS !(T2,0,ABRID !(!T1,1,PATHS !T1,1,ACTIV))
P1,0 A1*11 SACTIV

North-east over a BridS (Path reactivation and movement)
6
SPATHS N0*1* !(!E0,-1 !T0,-1,OBSTA) !(!E-1,0 !T-1,0,OBSTA) T1,0,BRISS E1,1
P1,1 A**11 SACTIV

North over a BridS (Path reactivation and movement)
6
SPATHS N0*1* !(!T1,1,PATHS !T1,1,BRID* !T1,1,BRIS*) T1,0,BRISS !(!E0,-1 !T0,-1,OBSTA) !(!E-1,0 !T-1,0,OBSTA) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV))
P0,1 A1*1* SACTIV

Bridge A
100
SACTIV !(!T0,1,PATHS !T0,1,BRID* !T0,1,BRIS*) !(!T0,-1,PATHS !T0,-1,DIAG* !T0,-1,BRIS*) !(!T1,1,PATHS !T1,1,BRID* !T1,1,BRIS*) !(!T1,-1,PATHS !T1,-1,BRID* !T1,-1,BRIS* !T1,-1,TRAS*) !(!E-1,1 !(T-1,1,PATHS T0,1,*BRID)) !E1,0 !T1,0,ACTIV !T1,0,TRAS* !(T1,0,*BRID T-1,0,*BRID) !(T0,1,*BRID T0,-1,*BRID) E-1,0
SABRID

North over Bridge A
5
SACTIV N0*1* T1,0,ABRID !(!T1,1,PATHS !T1,1,DIAG* !T1,1,BRID* !T1,1,BRIS* !T1,1,*BRID) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV)) !(T0,2,ACTIV E1,2)
P0,1 A1*1*

North over Bridge A II
4
SACTIV N0*1* T1,1,ABRID !(!T1,0,PATHS !T1,0,DIAG* !T1,0,BRID* !T1,0,*BRID !T1,0,BRIS*) !(T0,2,BBRID !(!T-1,1,PATHS !T-1,1,ACTIV)) !(T0,2,ACTIV E1,2)
P0,1 A1*1*

Bridge A movement

100
SABRID !(!T-1,1,PATHS !T-1,-1,BRIS*) !(!T-1,-1,PATHS !T-1,-1,BRIS*) E-1,0
P-1,0 A11*1

Bridge A reactivation
100
SABRID E0,-1 E-1,0
SACTIV

Bridge A reactivation II
100
SABRID E0,-1 !(!T-1,0,BBRID !T-1,0,PATHS) E-1,-1 !T1,-1,ACTIV
SACTIV P-1,-1

Module reactivation over a Bridge A
5
SPATHS T0,1,ABRID E-1,0 E-1,1 !(!T0,-1,OBSTA !E0,-1 !(T0,-1,PATHS T-1,-1,OBSTA))
SACTIV P-1,1 A**1*

Module reactivation over a Bridge A III
5
SPATHS T0,1,ABRID !(!T-1,1,BBRID !T-1,1,PATHS) E-1,0 !(!T0,-1,OBSTA !E0,-1) T1,-1,OBSTA
SACTIV P-1,0 A11*1

West over a bridge A
5
SACTIV T-1,1,ABRID T0,1,PATHS !(!E0,-1 !T0,-1,OBSTA) E-1,0 !(!T-2,1,BBRID !T-2,1,PATHS)
P-1,0 A11**

North-West over a bridge A
5
SACTIV N10*0 E-1,1 T0,1,ABRID
P-1,1 A1*1*

Bridge B
100
SACTIV !(!T1,0,PATHS !T1,0,DIAG* !T1,0,BRIS* !T1,0,BRID*) !(!T-1,0,PATHS !T-1,0,DIAG* !T-1,0,BRID* !T-1,0,BRIS*) !(!E-1,-1 !(T-1,-1,PATHS T-1,0,*BRID)) !(!T1,1,PATHS !T1,1,BRID* !T1,1,BRIS*) !(!T-1,1,PATHS !T-1,1,BRID* !T-1,1,BRIS*) !E0,1 !T0,1,ACTIV !T0,1,TRAS* !(T1,0,*BRID T-1,0,*BRID) !(T0,1,*BRID T0,-1,*BRID) E0,-1
SBBRID

West over a bridge B
5
SACTIV N10** T0,1,BBRID !(!T-1,1,PATHS !T-1,1,DIAG* !T-1,1,BRID* !T-1,1,BRIS* !T-1,-1,*BRID) !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 A11**

West over a bridge B II
5
SACTIV N10** T-1,1,BBRID !(!T0,1,PATHS !T0,1,DIAG* !T0,1,BRID* !T0,1,*BRID !T0,1,BRIS*) !(T-2,0,CBRID !(!T-1,-1,PATHS !T-1,-1,ACTIV))
P-1,0 A11**

South-west over a bridge B
5
SACTIV N11*0 T0,1,BBRID T-1,0,PATHS E-1,-1
P-1,-1 A11**

South-west II over a bridge B
5
SACTIV N11*0 T-1,0,BBRID E-1,-1
P-1,-1 A11**

South over a bridge B II
5
SACTIV N*1*0 T-1,-1,BBRID !(!T-1,-2,CBRID !T-1,-2,PATHS) T-1,0,PATHS !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV)) !(!E1,0 !T1,0,OBSTA)
P0,-1 A11**

South over a bridge B III
4
SACTIV N*1*0 T-1,0,BBRID T-1,-1,PATHS !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV)) !(!E1,0 !T1,0,OBSTA) !(!T-1,-1,CBRID !T-1,-1,PATHS)
P0,-1 A11**

Bridge B movement
100
SBBRID !(!T-1,-1,PATHS !T-1,-1,BRIS*) !(!T1,-1,PATHS !T1,-1,BRIS*) E0,-1
P0,-1 A*111

Bridge B reactivation
100
SBBRID E1,0 E0,-1
SACTIV

Bridge B reactivation II
100
SBBRID E1,0 !(!T0,-1,CBRID !T0,-1,PATHS) E1,-1 !T1,1,ACTIV
SACTIV P1,-1

Module reactivation over a Bridge B

5
SPATHS T-1,0,BBRID E0,-1 E-1,-1 !(!T1,0,OBSTA !E1,0 !(T1,0,PATHS T1,-1,OBSTA))
SACTIV P-1,-1 A**1*

Module reactivation over a Bridge B III
5
SPATHS T-1,0,BBRID !(!T-1,-1,CBRID !T-1,-1,PATHS) E0,-1 !(!T1,0,OBSTA !E1,0) T1,1,OBSTA
SACTIV P0,-1 A*11*

Bridge C
100
SACTIV !(!T0,1,PATHS !T0,1,TRAS* !T0,1,DIAG* !T0,1,BRID* !T0,1,BRIS*) !(!T0,-1,PATHS !T0,-1,DIAG* !T0,-1,BRID* !T0,-1,BRIS*) !(!E1,-1 !(T1,-1,PATHS T0,-1,*BRID)) !(!T-1,-1,PATHS !T-1,-1,BRID* !T-1,-1,BRIS*) !(!T-1,1,PATHS !T-1,1,BRID* !T-1,1,BRIS*) !E-1,0 !T-1,0,ACTIV !T-1,0,TRAS* !(T1,0,*BRID T-1,0,*BRID) !(T0,1,*BRID T0,-1,*BRID) E1,0
SCBRID

Bridge C Movement
100
SCBRID !(!T1,1,PATHS !T1,1,BRIS*) !(!T1,-1,PATHS !T1,-1,BRIS*) E1,0
P1,0 A1*11

Bridge C reactivation
100
SCBRID E0,1 E1,0
SACTIV

Bridge C reactivation II
100
SCBRID E0,1 !(!T1,0,DBRID !T1,0,PATHS) E1,1 !T-1,1,ACTIV
SACTIV P1,1

Module reactivation over a Bridge C
5
SPATHS T0,-1,CBRID E1,0 E1,-1 !(!T0,1,OBSTA !E0,1 !(T0,1,PATHS T1,1,OBSTA))
SACTIV P1,-1 A**11

Module reactivation over a Bridge C III
5
SPATHS T0,-1,CBRID !(!T1,-1,DBRID !T1,-1,PATHS) E1,0 T-1,1,OBSTA !(!T0,1,OBSTA !E0,1)
SACTIV P1,0 A**11

South over a Bridge C
7
SACTIV N*1*0 T-1,0,CBRID !(!T-1,-1,PATHS !T-1,-1,DIAG* !T-1,-1,BRID* !T-1,-1,*BRID !T-1,-1,BRIS*) !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV))
P0,-1 A11**

South over a Bridge C II
7
SACTIV N*1*0 T-1,-1,CBRID !(!T-1,0,PATHS !T-1,0,DIAG* !T-1,0,BRID* !T-1,0,BRIS* !T-1,0,*BRID) !(T0,-2,DBRID !(!T1,-1,PATHS !T1,-1,ACTIV))
P0,-1 A11**

East over a Bridge C
6
SACTIV T1,-1,CBRID !(!T2,-1,PATHS !T2,-1,DBRID) !(!T0,1,OBSTA !E0,1) T-1,1,OBSTA
P1,0 A**01

South-East over a Bridge C
6
SACTIV T0,-1,CBRID T1,-2,PATHS E1,0 E1,-1
P1,-1 A***1

Bridge D formation
100
SACTIV !(!T-1,0,PATHS !T-1,0,DIAG* !T-1,0,BRID* !T-1,0,BRIS*) !(!T1,0,PATHS !T1,0,DIAG* !T1,0,BRID* !T1,0,BRIS*) !(!T1,-1,PATHS !T1,-1,BRID* !T1,-1,BRIS*) !(!T-1,-1,PATHS !T-1,-1,BRID* !T-1,-1,BRIS*) !(!E1,1 !(T1,1,PATHS T1,0,*BRID)) !T0,-1,ACTIV !E0,-1 !T0,-1,TRAS* !(T1,0,*BRID T-1,0,*BRID) !(T0,1,*BRID T0,-1,*BRID) E0,1
SDBRID

Bridge D movement
100
SDBRID !(!T1,1,PATHS !T1,1,BRIS*) !(!T-1,1,PATHS !T-1,1,BRIS*) E0,1
P0,1 A111*

Bridge D reactivation
100
SDBRID E-1,0 E0,1
SACTIV

Bridge D reactivation II
100
SDBRID !(!T0,1,ABRID !T0,1,PATHS) E-1,1 E-1,0 !T-1,-1,ACTIV
SACTIV P-1,1

Module reactivation over a Bridge D
5
SPATHS T1,0,DBRID E0,1 E1,1 !(!T-1,0,OBSTA !E-1,0 !(T-1,0,PATHS T-1,1,OBSTA))
SACTIV P1,1 A**11

Module reactivation over a Bridge D III
5
SPATHS T1,0,DBRID !(!T1,1,ABRID !T1,1,PATHS) E0,1 !(!T-1,0,OBSTA !E-1,0) T-1,-1,OBSTA
SACTIV P0,1 A**11

East over a Bridge D
8
SACTIV N**01 T0,-1,DBRID !(!T1,-1,PATHS !T1,-1,DIAG* !T1,-1,BRID* !T1,-1,*BRID !T1,-1,BRIS*) !(T2,0,ABRID !(!T1,1,PATHS !T1,1,ACTIV)) !(T2,0,ACTIV E2,-1) !(T2,1,ACTIV E2,0 T1,1,PATHS)
P1,0 A11**

East over a Bridge D II
6
SACTIV N**01 T1,-1,DBRID !(!T0,-1,PATHS !T0,-1,DIAG* !T0,-1,BRID* !T0,-1,*BRID !T0,-1,BRIS*) !(T2,0,ABRID !(!T1,1,PATHS !T1,1,ACTIV)) !(T2,0,ACTIV E2,-1) !(T2,1,ACTIV E2,0 T1,1,PATHS)
P1,0 A11**

North over a Bridge D
6
SACTIV T1,1,DBRID !(!T1,2,ABRID !T1,2,PATHS) !(!E-1,0 !T-1,0,OBSTA) E0,1 !(!T0,-1,OBSTA !E0,-1)
P0,1 A**1*

North over a Bridge D II
6
SACTIV T1,0,DBRID !(!T1,1,ABRID !T1,1,PATHS) !(!E-1,0 !T-1,0,OBSTA) E0,1 !(!T0,-1,OBSTA !E0,-1)
P0,1 A**1*

North-East over a Bridge D
6
SACTIV N0**1 T1,0,DBRID E1,1
P1,1 A1*11

Bridge F
100

SACTIV !(!E1,0 !T1,0,ACTIV) T0,1,PATHS (!V0,1,C000 0001 W0,1,C000 0001) T0,-1,PATHS E-1,1 !(!E1,1 !T1,1,ACTIV !T1,1,*BRID !T1,1,BRID* !T1,1,BRIS*) !(!E1,-1 !T1,-1,ACTIV !T1,-1,*BRID !T1,-1,BRID* !T1,-1,BRIS*) SBRIDF

Bridge F II
100
SACTIV T1,0,BRIDF T0,-1,PATHS T0,1,PATHS
SBRIDF

Bridge F reactivation
15
SBRIDF E0,-1 E-1,0
SACTIV

Path reactivation over a bridge F
15
SPATHS T0,1,BRIDF E-1,0 E-1,1 !(!E0,-1 !T0,-1,OBSTA !(T0,-1,PATHS T-1,-1,OBSTA))
SACTIV P-1,1 A**1*

North over a bridge F
6
SACTIV N0*1* T1,0,BRIDF T1,1,PATHS
P0,1 A1***

North over a bridge F II
6
SACTIV N0*1* T1,1,BRIDF T1,0,PATHS
P0,1 A1***

Bridge G
100
SACTIV !(!E0,1 !T0,1,ACTIV) T-1,0,PATHS (!V-1,0,C000 0001 W-1,0,C000 0001) T1,0,PATHS E-1,-1 !(!E-1,1 !T-1,1,ACTIV !T-1,1,CBRID !T-1,1,BRISO) !(!E1,1 !T1,1,ACTIV !T1,1,ABRID !T1,1,BRISE) SBRIDG

Bridge G II
100
SACTIV T0,1,BRIDG T-1,0,PATHS T1,0,PATHS
SBRIDG

Bridge G reactivation
15

SBRIDG E1,0
SACTIV

Path reactivation over a bridge G
15
SPATHS T-1,0,BRIDG E0,-1 E-1,-1 !(!E1,0 !T1,0,OBSTA !(T1,0,PATHS T1,-1,OBSTA))
SACTIV P-1,-1 A1***

North-west over a bridge G
6
SACTIV N10*0 E-1,1 T0,1,BRIDG
P-1,1 A1*1*

West over a bridge G
6
SACTIV N10*0 T-1,1,BRIDG T0,1,PATHS E-1,0
P-1,0 A1***

South-west over a bridge G
6
SACTIV T-1,0,PATHS T0,1,BRIDG E0,-1 E-1,-1
P-1,-1 A1***

Bridge H
100
SACTIV !(!E-1,0 !T-1,0,ACTIV) T0,1,PATHS T0,-1,PATHS (!V0,-1,C000 0001 W0,-1,C000 0001) E1,-1 !(!E-1,1 !T-1,1,ACTIV !T-1,1,BBRID !T-1,1,BRISN) !(!E-1,-1 !T-1,-1,ACTIV !T-1,-1,DBRID !T-1,-1,BRISS) SBRIDH

Bridge H II
100
SACTIV T-1,0,BRIDH T0,-1,PATHS T0,1,PATHS
SBRIDH

Bridge H reactivation
15
SBRIDH E0,1 E1,0
SACTIV

Path module over a bridge H
15
SPATHS T0,-1,BRIDH E1,0 E1,-1 !(!E0,1 !T0,1,OBSTA !(T0,1,PATHS T1,1,OBSTA))
SACTIV P1,-1 A**1*

South over a bridge H
6
SACTIV N*1*0 T-1,0,BRIDH T-1,-1,PATHS !(!T1,0,ACTIV !E1,0)
P0,-1 A*1*1

South over a bridge H II
6
SACTIV N*100 T-1,-1,BRIDH T-1,0,PATHS
P0,-1 A*1*1

Bridge K
100
SACTIV !(!E0,-1 !T0,-1,ACTIV) T-1,0,PATHS T1,0,PATHS (!V1,0,C000 0001
W1,0,C000 0001) E1,1 !(!E-1,-1 !T-1,-1,ACTIV !T-1,-1,CBRID !T-1,-1,PATHS)
!(!E1,-1 !T1,-1,ACTIV !T1,-1,ABRID !T1,-1,TRASE !T1,-1,PATHS) SBRIDK

Bridge K II
100
SACTIV T0,-1,BRIDK T-1,0,PATHS T1,0,PATHS
SBRIDK

Bridge K reactivation
15
SBRIDK E-1,0 E0,1
SACTIV

Path reactivation over a bridge K
15
SPATHS T1,0,BRIDK E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA !(T-1,0,PATHS T-1,1,OBSTA))
SACTIV P1,1 A1***

East over a bridge K
6
SACTIV N0*01 T1,-1,BRIDK T0,-1,PATHS E1,0
P1,0 A*1*1

East over a bridge K II
6
SACTIV N0*01 T1,-1,PATHS T0,-1,BRIDK E1,0
P1,0 A*1*1

Bridge M
100
SACTIV E0,-1 !(!E0,1 !T0,1,ACTIV) T-1,0,PATHS (!V-1,0,C000 0001 W-1,0,C000

0001) T1,0,PATHS E-1,-1 !(!E-1,1 !T-1,1,ACTIV !T-1,1,CBRID) T1,1,PATHS SBRIDM

Bridge M II
100
SACTIV !(!T0,1,BRIDM !T0,1,TRAS*) T-1,0,PATHS T1,0,PATHS SBRIDM

Bridge M III
100
SACTIV !(!E0,1 !T0,1,ACTIV) T-1,0,PATHS (!V-1,0,C000 0001 W-1,0,C000 0001) T1,0,PATHS E-1,-1 !(!E1,1 !T1,1,ACTIV !T1,1,ABRID !T1,1,BRISE) T-1,1,PATHS SBRIDM

Bridge M reactivation
15
SBRIDM E1,0
SACTIV

Reactivation over a bridge M
15
SPATHS T-1,0,BRIDM E0,-1 E-1,-1 !(!E1,0 !T1,0,OBSTA !(T1,0,PATHS T1,-1,OBSTA))
SACTIV P-1,-1 A1***

North-west over a bridge M
6
SACTIV N10*0 E-1,1 T0,1,BRIDM
P-1,1 A1*1*

West over a bridge M
6
SACTIV N10** T-1,1,BRIDM T0,1,PATHS E-1,0 !(!T0,-1,OBSTA !E0,-1)
P-1,0 A1***

South-west over a bridge M
6
SACTIV T-1,0,PATHS T0,1,BRIDM E0,-1 E-1,-1
P-1,-1 A1***

Bridge P
100

SACTIV !(!E0,-1 !T0,-1,ACTIV !T0,-1,TRAS*) T-1,0,PATHS T1,0,PATHS (!V1,0,C000 0001 W1,0,C000 0001) (!V1,0,C001 0001 W1,0,C000 0001) E1,1 !(!T1,-1,PATHS !T1,-1,*BRID !T1,-1,BRID*) !(!E-1,-1 !T-1,-1,ACTIV !T-1,-1,*BRID !T-1,-1,BRID*) SBRIDP

Bridge P II
100
SACTIV !(!T0,-1,BRIDP !T0,-1,TRAS*) T-1,0,PATHS T1,0,PATHS
SBRIDP

Bridge P III
100
SACTIV !(!E0,-1 !T0,-1,ACTIV !T0,-1,TRAS*) T-1,0,PATHS T1,0,PATHS (!V1,0,C000 0001 W1,0,C000 0001) E1,1 T-1,-1,PATHS !(!E1,-1 !T1,-1,ACTIV !T1,-1,ABRID !T-1,-1,CBRID) SBRIDP

Bridge P reactivation
15
SBRIDP E-1,0 E0,1
SACTIV

Path reactivation over a bridge P
15
SPATHS T1,0,BRIDP E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA !(T-1,0,PATHS T-1,1,OBSTA))
SACTIV P1,1 A1***

East over a bridge P
6
SACTIV N**01 T1,-1,BRIDP T0,-1,PATHS E1,0 !(!T0,1,ACTIV !E0,1)
P1,0 A*1*1

East sover a bridge P II
6
SACTIV N**01 T1,-1,PATHS T0,-1,BRIDP E1,0 !(!T0,1,ACTIV !E0,1)
P1,0 A*1*1

Bridge Q
100
SACTIV !(!E-1,0 !T-1,0,ACTIV !T-1,0,TRAS*) T0,1,PATHS T0,-1,PATHS (!V0,-1,C000 0001 W0,-1,C000 0001) !(!E1,-1 !T1,-1,TRAS*) !(!E-1,-1 !T-1,-1,ACTIV !T-1,-1,*BRID !T-1,-1,BRID* !T-1,-1,BRIS*) T-1,1,PATHS
SBRIDQ

Bridge Q II
100
SACTIV !(!T-1,0,BRIDQ !T-1,0,TRAS*) T0,-1,PATHS T0,1,PATHS
SBRIDQ

Bridge Q III

100
SACTIV !(!E-1,0 !T-1,0,ACTIV !T-1,0,TRAS*) T0,1,PATHS T0,-1,PATHS (!V0,-1,C000 0001 W0,-1,C000 0001) E1,-1 !(!E-1,1 !T-1,1,ACTIV !T-1,1,ABRID !T-1,1,CBRID) T-1,-1,PATHS SBRIDQ

Bridge Q reactivation
15
SBRIDQ E0,1 E1,0
SACTIV

Path reactivation over a bridge Q
15
SPATHS T0,-1,BRIDQ E1,0 E1,-1 !(!E0,1 !T0,1,OBSTA !(T0,1,PATHS T1,1,OBSTA))
SACTIV P1,-1 A*11*

South over a bridge Q
6
SACTIV N***0 T-1,0,BRIDQ T-1,-1,PATHS !(!T1,0,ACTIV !E1,0)
P0,-1 A*1*1

South over a bridge Q II
6
SACTIV N**00 T-1,-1,BRIDQ T-1,0,PATHS
P0,-1 A*1*1

Bridge R
100
SACTIV !(!E1,0 !T1,0,ACTIV !T1,0,TRAS*) T0,1,PATHS (!V0,1,C000 0001 W0,1,C000 0001) T0,-1,PATHS E-1,1 !(!E1,1 !T1,1,ACTIV !T1,1,*BRID !T1,1,BRID* !T1,1,BRIS*) T1,-1,PATHS SBRIDR

Bridge R II
100
SACTIV !(!T1,0,BRIDR !T1,0,TRAS*) T0,-1,PATHS T0,1,PATHS
SBRIDR

Bridge R III
100
SACTIV !(!E-1,0 !T-1,0,ACTIV !T-1,0,TRAS*) T0,1,PATHS (!V0,1,C000 0001 W0,1,C000 0001) T0,-1,PATHS E-1,1 !(!E1,-1 !T1,-1,ACTIV !T1,-1,*BRID) T1,1,PATHS
SBRIDR

Bridge R reactivation
15
SBRIDR E0,-1 E-1,0
SACTIV

Path reactivation over a bridge R
15
SPATHS T0,1,BRIDR E-1,0 E-1,1 !(!E0,-1 !T0,-1,OBSTA !(T0,-1,PATHS T-1,-1,OBSTA))
SACTIV P-1,1 A*11*

North over a bridge R
6
SACTIV N0*1* T1,0,BRIDR T1,1,PATHS !(!T-1,0,ACTIV !E-1,0 !T-1,0,PATHS)
P0,1 A*1*1

North over a bridge R II
6
SACTIV N0*1* T1,0,PATHS T1,1,BRIDR !(!T-1,0,ACTIV !E-1,0)
P0,1 A**1*

Bridge W
100
SACTIV !(!E1,0 !T1,0,ACTIV !T1,0,TRAS*) T0,1,PATHS T0,-1,PATHS E-1,1!(!E-1,-1 !T-1,-1,ACTIV) !(!E1,1 !T1,1,ACTIV !T1,1,ABRID !T1,1,CBRID) T1,-1,PATHS
SBRIDW

Bridge W reactivation
15
SBRIDW E0,-1 E-1,0
SACTIV

Path reactivation over a bridge W
15
SPATHS T0,1,BRIDW E-1,0 E-1,1 !(!E0,-1 !T0,-1,OBSTA !(T0,-1,PATHS T-1,-1,OBSTA))
SACTIV P-1,1 A*11*

North over a bridge W
6
SACTIV N001* T1,0,BRIDW T1,1,PATHS
P0,1 A*1*1

North over a bridge W II
6
SACTIV N001* T1,0,PATHS T1,1,BRIDW
P0,1 A**1*

Bridge Z
100

SACTIV !(!E0,1 !T0,1,ACTIV) T-1,0,PATHS T1,0,PATHS E-1,-1 !(!E-1,1 !T-1,1,ACTIV !T-1,1,CBRID) T1,1,PATHS !(!E1,-1 !T1,-1,ACTIV)
SBRIDZ

Bridge Z reactivation
15
SBRIDZ E1,0
SACTIV

Path reactivation over a bridge Z
15
SPATHS T-1,0,BRIDZ E0,-1 E-1,-1 !(!E1,0 !T1,0,OBSTA !(T1,0,PATHS T1,-1,OBSTA))
SACTIV P-1,-1 A1***

North-east over a bridge Z
6
SACTIV N10*0 E-1,1 T0,1,BRIDZ
P-1,1 A1*1*

West over a brige Z
6
SACTIV N10*0 T-1,1,BRIDZ T0,1,PATHS E-1,0
P-1,0 A1***

South-west over a bridge Z
6
SACTIV T-1,0,PATHS T0,1,BRIDZ E0,-1 E-1,-1
P-1,-1 A1***

Bridge J
100
SACTIV !(!E0,-1 !T0,-1,ACTIV !T0,-1,TRAS*) T-1,0,PATHS T1,0,PATHS E1,1 T-1,-1,PATHS !(!E1,-1 !T1,-1,ACTIV !T1,-1,*BRID) !(!E-1,1 !T-1,1,ACTIV)
SBRIDJ

Bridge J reactivation
15
SBRIDJ E-1,0 E0,1
SACTIV

Path reactivation over a bridge J
15
SPATHS T1,0,BRIDJ E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA !(T-1,0,PATHS T-1,1,OBSTA))
SACTIV P1,1 A1***

East over a bridge J
6
SACTIV N0*01 T1,-1,BRIDJ T0,-1,PATHS E1,0
P1,0 A*1*1

East over a bridge J II
6
SACTIV N0*01 T1,-1,PATHS T0,-1,BRIDJ E1,0
P1,0 A*1*1

Bridge Y
100
SACTIV !(!E-1,0 !T-1,0,ACTIV !T-1,0,TRAS*) T0,1,PATHS T0,-1,PATHS E1,-
1 !(!E1,1 !T1,1,ACTIV) !(!E-1,-1 !T-1,-1,ACTIV !T-1,-1,*BRID !T-1,-1,BRISS)
T-1,1,PATHS
SBRIDY

Bridge Y reactivation
15
SBRIDY E0,1 E1,0
SACTIV

Path reactivation over a bridge Y
15
SPATHS T0,-1,BRIDY E1,0 E1,-1 !(!E0,1 !T0,1,OBSTA !(T0,1,PATHS T1,1,OBSTA))
SACTIV P1,-1 A*11*

South over a bridge Y
6
SACTIV N**00 T-1,0,BRIDY T-1,-1,PATHS
P0,-1 A*1*1

South over a bridge Y II
6
SACTIV N**00 T-1,-1,BRIDY T-1,0,PATHS
P0,-1 A*1*1

Transversal bridge A
100
SACTIV N*01* T1,0,PATHS T-1,1,PATHS !(!T0,1,ACTIV !E0,1) !(!T0,-1,ACTIV
!E0,-1) !T0,2,*BRID !T0,2,BRID* !T0,2,BRIS* !(!T1,1,ACTIV !E1,1) !V-1,1,C000
0001 W-1,1,C000 0001 !E1,-1
STRASA

Transversal bridge A reactivation
100
STRASA E1,0 T-1,0,PATHS E0,-1
SACTIV

North-west over a transversal bridge A
7
SACTIV N10** T0,1,TRASA E-1,1
P-1,1 A111* SPATHS C002 + 0000 0001 C000 + 0000 0001

West over a transversal bridge A
6
SACTIV N10** !(!T-1,1,TRASA !T0,1,TRASA)
P-1,0 A11*1

Path reactivation over a transversal bridge A
15
SPATHS N*110 T-1,0,TRASA T1,0,PATHS E0,-1 E-1,-1
SACTIV P-1,-1 A11**

Transversal bridge B
100
SACTIV N*10* T-1,0,PATHS T1,-1,PATHS !(!T0,-1,ACTIV !E0,-1) !(!T0,1,ACTIV !E0,1) !T0,-2,*BRID !T0,-2,BRID* !(!T-1,-1,ACTIV !E-1,-1) !V1,-1,C000 0001 W1,-1,C000 0001 !E-1,-1
STRASB

Transversal bridge B Reactivation
100
STRASB E-1,0 T1,0,PATHS E0,1
SACTIV

South-east over a transversal bridge B
7
SACTIV N0*01 T0,-1,TRASB E1,-1
P1,-1 A111* SPATHS C002 + 0000 0001 C000 + 0000 0001

East over a transversal bridge B
6
SACTIV N0*01 !(!T1,-1,TRASB !T0,-1,TRASB)
P1,0 A**11

Path reactivation over a transversal bridge B
15
SPATHS N011* T1,0,TRASB T-1,0,PATHS E0,1 E1,1
SACTIV P1,1 A1*11

Transversal bridge C
100
SACTIV N1*0* !(!T-1,0,PATHS !E-1,0 !T-1,0,ACTIV) T0,1,PATHS T1,-1,PATHS
!(!T1,1,ACTIV !E1,1) !(!E0,-1 !T0,-1,ACTIV) !V1,-1,C000 0001 W1,-1,C000 0001
STRASC

Transversal bridge C reactivation
100
STRASC E0,1 T1,0,PATHS E1,1 !(!T-1,0,PATHS !E-1,0) E-1,1
SACTIV P1,1

South over a transversal bridge C
6
SACTIV N*100 !(!T-1,-1,TRASC !T-1,0,TRASC)
P0,-1 A111* SACTIV

North-east over a transversal bridge C
6
SACTIV N0111 T1,0,TRASC E1,1 T-1,0,OBSTA
P1,1 A111* SACTIV

East over transversal bridge C
6
SACTIV N**01 !(!T1,-1,TRASC !T0,-1,TRASC) !(!E0,1 !T0,1,OBSTA) !(!E-1,0
!T-1,0,OBSTA)
P1,0 A111* SACTIV

Path formation over a transversal bridge C
10
SACTIV T-1,0,TRASC T0,-1,PATHS
SPATHS C002 + 0000 0001 C000 + 0000 0001

Path reactivation over a transversal bridge C
15
SPATHS N0*01 T0,-1,TRASC E1,0 E1,-1
SACTIV P1,-1 A1*1*

Path reactivation over a transversal bridge II C
15

SPATHS N0*01 T0,-1,TRASC E1,0 T1,-1,PATHS
SACTIV P1,0 A1*1*

Transversal bridge D
100
SACTIV N*0*1 !(!T1,0,PATHS !E1,0 !T1,0,ACTIV) T0,-1,PATHS T-1,1,PATHS !(!T-1,-1,ACTIV !E-1,-1) !(!E0,1 !T0,1,ACTIV) !V-1,1,C000 0001 W-1,1,C000 0001 !E0,-2
STRASD

Reactivation of a transversal bridge D
100
STRASD T-1,0,PATHS !(!T1,0,PATHS !T0,1,ACTIV !E0,1) E-1,-1 E0,-1 E1,-1
SACTIV P-1,-1

North over a transversal bridge D
6
SACTIV N0*1* !(!T1,1,TRASD !T1,0,TRASD) !(!E-1,0 !T-1,0,ACTIV)
P0,1 A111* SACTIV

Path formation over transversal D
10
SACTIV T1,0,TRASD T0,1,PATHS
SPATHS C002 + 0000 0001 C000 + 0000 0001

Module activation over transversal D
15
SPATHS N10** T0,1,TRASD E-1,0 T-1,1,PATHS !(!T0,-1,OBSTA !E0,-1)
SACTIV A1*1*

West over a transversal D
15
SACTIV N10** T0,1,TRASD T-1,1,PATHS !(!T0,-1,OBSTA !E0,-1)
P-1,0 A1*1*

West over a transversal D II
15
SACTIV N10** T-1,1,TRASD T0,1,PATHS !(!T0,-1,OBSTA !E0,-1)
SACTIV P-1,0 A1*1*

Transversal bridge E
100
SACTIV N***1 !(!T-1,0,ACTIV !E-1,0) T0,-1,PATHS T1,1,PATHS !(!T1,-1,ACTIV !E1,-1) !(!E1,0 !T1,0,ACTIV) !V1,1,C000 0001 W1,1,C000 0001 !E-1,-1 STRASE

Transversal bridge E reactivation
100
STRASE E0,-1 T0,1,PATHS E-1,0
SACTIV

North over a transversal bridge E
6
SACTIV N0*1* !(!T1,1,TRASE !T1,0,TRASE) !E1,1
P0,1 A111* SACTIV

North-east over a transversal bridge E
6
SACTIV N0*1* T1,0,TRASE E1,1
P1,1 A111* SPATHS C002 + 0000 0001 C000 + 0000 0001

Path reactivation over a transversal bridge E
15
SPATHS N10*1 T0,-1,PATHS T0,1,TRASE E-1,0 E-1,1
SACTIV P-1,1 A1*1*

Transversal bridge F
100
SACTIV N1*** T0,1,PATHS !(!T-1,0,ACTIV !E-1,0) !(!T1,0,ACTIV !E1,0) T-1,-1,PATHS !(!T-1,1,ACTIV !E-1,1) !(!E0,-1 !T0,-1,ACTIV) !V-1,-1,C000 0001 W-1,-1,C000 0001 !E1,1
STRASF

Transversal bridge F reactivation
100
STRASF E0,1 T0,-1,PATHS E1,0
SACTIV

South over a transversal bridge F
6
SACTIV N*1*0 !(!T-1,-1,TRASF !T-1,0,TRASF) !E-1,-1
P0,-1 A11*1 SACTIV

South-west over a transversal bridge F
6
SACTIV N*1*0 T-1,0,TRASF E-1,-1
P-1,-1 A11*1 SPATHS C002 + 0000 0001 C000 + 0000 0001

Path reactivation over a transversal bridge F
15
SPATHS N**01 T0,-1,TRASF T0,1,PATHS E1,0 E1,-1
SACTIV P1,-1 A1*1*

Transversal bridge G
100
SACTIV N**10 !(!E0,1 !T0,1,PATHS) T1,0,PATHS T-1,-1,PATHS !(!T1,-1,ACTIV
!E1,-1) !(!E-1,0 !T-1,0,ACTIV) !V-1,-1,C000 0001 W-1,-1,C000 0001
STRASG

Transversal bridge G reactivation
100
STRASG E1,0 T-1,0,PATHS !(!E0,-1 !T0,-1,ACTIV)
SACTIV

Transversal bridge G reactivation II
100
STRASG E1,0 T-1,0,PATHS T0,-1,PATHS E1,-1
SACTIV

Path formation over a transversal bridge G
100
SACTIV T1,0,TRASG T0,-1,PATHS
SPATHS C000 + 0000 0001 C002 + 0000 0001

South-west over a transversal bridge G
6
SACTIV T0,1,TRASG E-1,-1 T-1,0,PATHS
P-1,-1 A11**

West over a transversal bridge G
6
SACTIV N10** !(!T-1,1,TRASG !T0,1,TRASG) !E-1,1
P-1,0 A11*1

Path reactivation over a transversal bridge G
15
SPATHS N*100 T-1,0,TRASG T0,1,PATHS E-1,-1
SACTIV P-1,-1 A11**

Path reactivation over a transversal bridge G II
15
SPATHS N*100 T-1,0,TRASG T0,1,PATHS !(!T-1,-1,CBRID !T-1,-1,PATHS)
SACTIV P0,-1 A11**

Path reactivation over a transversal bridge G III
15
SPATHS N0001 T0,-1,PATHS E1,-1
SACTIV P1,-1 A111*

Transversal bridge H
100
SACTIV N01** !(!T0,-1,PATHS !E0,-1) T-1,0,PATHS T1,1,PATHS !(!T1,0,ACTIV
!E1,0) !(!E-1,1 !T-1,1,ACTIV) !V1,1,C000 0001 W1,1,C000 0001
STRASH

North over a transversal bridge H
100
SACTIV E0,1 T1,1,TRASH !(!E-1,0 !T-1,0,OBSTA) E-1,1
P0,1 SACTIV

North over a transversal bridge H II
100
SACTIV T1,0,TRASH !(!E-1,0 !T-1,0,OBSTA) E0,1
P0,1 SACTIV

Transversal bridge H reactivation
100
STRASH E-1,0 T0,1,PATHS E-1,1 E-1,-1
SACTIV

Path formation over a transversal bridge H
100
SACTIV T0,-1,TRASH T1,0,PATHS
SPATHS C000 + 0000 0001 C002 + 0000 0001

East over a transversal bridge H
6
SACTIV N**01 !(!T1,-1,TRASH !T0,-1,TRASH) !E1,-1
P1,0 A11*1

Path reactivation over a transversal bridge H
15
SPATHS N0*11 T1,0,TRASH !(!T-1,0,OBSTA !E-1,0)
SACTIV P0,1 A**1*

Diagonal A
100
SACTIV N*100 T-1,0,PATHS T-1,1,PATHS T1,-2,PATHS E-1,-1 !V1,-2,C000 0001 W1,-2,C000 0001 !T2,-1,OBSTA !T2,-2,OBSTA
SDIAGA

Diagonal B
100
SACTIV N*001 T0,-1,PATHS T-1,1,DIAGA !(!T0,1,ACTIV !E0,1)
SDIAGB

East over diagonal A
6
SACTIV N*101 T1,-1,DIAGA
P1,0 A***1

South-east over diagonal A and B
6
SACTIV N**01 !(!T0,-1,DIAGA !T0,-1,DIAGB) !(!T-1,0,PATHS !T-1,0,DIAGA !E-1,0) E1,-1
P1,-1 A*1*1

South over diagonal B
6
SACTIV N*100 T-1,0,DIAGB T-1,-1,PATHS
P0,-1 A*101 SPATHS C000 + 0000 0001

Diagonal A reactivation
100
SDIAGA N0100 E-1,1 E-1,-1
SACTIV P-1,-1

Diagonal B reactivation
100
SDIAGB N0001 !T-1,1,DIAGA
SACTIV P1,0

Diagonal B reactivation II
100
SDIAGB N0011 !T-1,1,DIAGA T1,0,DBRID E1,1
SACTIV P1,1

Diagonal C
100

SACTIV N0*01 T-1,-1,PATHS T0,-1,PATHS T2,1,PATHS E1,-1 !V2,1,C000 0001 W2,1,C000 0001 !T1,2,OBSTA !T2,2,OBSTASDIAGC

Diagonal D
100
SACTIV N0*10 T1,0,PATHS T-1,-1,DIAGC !(!T-1,0,ACTIV !E-1,0)
SDIAGD

North-East over diagonal C e D
6
SACTIV N0*1* !(!T1,0,DIAGC !T1,0,DIAGD) E1,1
P1,1 A**11

North over diagonal C
6
SACTIV N0011 T1,1,DIAGC T1,0,PATHS
P0,1 A**1*

East over diagonal C
6
SACTIV N0*01 T0,-1,DBRID T1,-1,DIAGC
P1,0 A**1*

Diagonal C reactivation
100
SDIAGC N0001 T0,-1,PATHS E-1,-1 E1,-1
SACTIV P1,-1 A*1**

Diagonal D reactivation
100
SDIAGD N0010 !T-1,-1,DIAGC
SACTIV P0,1 A**1*

Diagonal D reactivation
100
SDIAGD N1010 !T-1,-1,DIAGC T0,1,*BRID E-1,1
SACTIV P-1,1 A**1*

Diagonal E
100
SACTIV N10*0 T1,1,PATHS !T-2,-2,OBSTA T0,1,PATHS !(!T-1,0,ACTIV !E-1,0) T-2,-1,PATHS E-1,1 !V-2,-1,C000 0001 W-2,-1,C000 0001 !T-1,-2,OBSTA !T1,0,OBSTA SDIAGE

Diagonal F
100

SACTIV N01*0 T-1,0,PATHS T1,1,DIAGE !(!T1,0,ACTIV !E1,0)
SDIAGF

South-west over diagonal E e F
6
SACTIV N*100 !(!T-1,0,DIAGE !T-1,0,DIAGF) !(!T0,1,PATHS !T0,1,DIAGE !E0,1) E-1,-1
P-1,-1 A*1*1

South-west over diagonal F II
6
SACTIV N1100 T0,1,DIAGF T-1,0,PATHS E-1,-1
P-1,-1 SPATHS C000 + 0000 0001 A11**

South over diagonal E
6
SACTIV N1100 T-1,0,PATHS T-1,-1,DIAGE
P0,-1 A*1**

Diagonal E reactivation
100
SDIAGE N1000 E1,1 E-1,1
SACTIV P-1,1 A**1*

Diagonal F reactivation
100
SDIAGF N0100 T-1,0,PATHS !T1,1,DIAGE
SACTIV P0,-1 A*1**

Diagonal F reactivation II
100
SDIAGF N010* T-1,0,PATHS !T1,1,DIAGE T0,-1,CBRID E1,-1
SACTIV P1,-1 A*1**

Diagonal G
100
SACTIV N001* !T-2,2,OBSTA T1,0,PATHS T1,-1,PATHS !(!T0,-1,ACTIV !E0,-1) T-1,2,PATHS E1,1 !V-1,2,C000 0001 W-1,2,C000 0001 !T-2,1,OBSTA
SDIAGG

Diagonal H
100
SACTIV N100* T0,1,PATHS T1,-1,DIAGG !(!T0,-1,ACTIV !E0,-1)
SDIAGH

North-west over diagonal G e H
6
SACTIV N10** !(!T0,1,DIAGG !T0,1,DIAGH) !(!T1,0,PATHS !T1,0,DIAGG !E1,0) E-1,1 !(!T0,-1,ACTIV !E0,-1 !(T0,-1,PATHS T1,0,PATHS T0,1,DIAGG))
P-1,1 A1*1*

North over diagonal G
6
SACTIV N001* T1,1,DIAGG T1,0,PATHS !(!T0,-1,ACTIV !E0,-1)
P0,1 A1*1*

North over diagonal H
6
SACTIV N0010 T1,0,DIAGH T1,1,PATHS
P0,1 A1*1*

West over diagonal G
6
SACTIV N101* T0,1,PATHS T-1,1,DIAGG !(!E0,-1 !T0,-1,OBSTA)
P-1,0 A1***

Diagonal G reactivation
100
SDIAGG N0010 E1,-1 E1,1
SACTIV P1,1 A**11

Diagonal H reactivation
100
SDIAGH N1000 T-1,1,PATHS !T1,-1,DIAGG
SACTIV P-1,0 A1***

Diagonal H reactivation II
100
SDIAGH N1100 T-1,0,*BRID !T1,-1,DIAGG E-1,-1
SACTIV P-1,-1 A11**

Auxiliary path reactivation 1
100
SPATHS N0100 T-1,0,PATHS !T-1,1,ACTIV !T-1,-1,ACTIV = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation 2
100

SPATHS N0010 T1,0,PATHS !T1,1,ACTIV !T1,-1,ACTIV = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation 3
100
SPATHS N1000 T0,1,PATHS !T1,1,ACTIV !T-1,1,ACTIV = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation 4
100
SPATHS N0001 T0,-1,PATHS !T1,-1,ACTIV !T-1,-1,ACTIV = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation - corner 1
100
SPATHS N1100 T-1,0,PATHS T0,1,PATHS T1,1,PATHS = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation - corner 2
100
SPATHS N0101 T-1,0,PATHS T0,-1,PATHS T-1,-1,PATHS = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation - corner 3
100
SPATHS N0011 T1,0,PATHS T0,-1,PATHS T1,-1,PATHS = C002 0001
SACTIV C002 + 0000 0000

Auxiliary path reactivation - corner 4
100
SPATHS N1010 T1,0,PATHS T0,1,PATHS T1,1,PATHS = C002 0001
SACTIV C002 + 0000 0000

Chapter 11

Conclusions

11.1 Presented results

The principal aim of this work was to propose a set of distributed algorithms for the in-shape locomotion of a 2-dimensional rectangular system, on free ground and in the presence of obstacles, specific for a parallel evaluation of all the rules of the algorithm to each module of the system at each step. This goal has been achieved, and the results presented go beyond the principal goal, as they have been extended to the case of 2-dimensional systems configured as histograms. Each algorithm presented is accompanied not only with a correctness proof and a complexity analysis, but with a direct implementation in a simulator, that allows the direct observation of the effect of the rules over different systems and obstacles. The algorithms presented are quite efficient, an the total number of time steps of the system analyzed in the different cases is proportional to the number of moves per module, as the rules are run in parallel. Although the sets of rules presented are conceived for a synchronous execution, an extension to an asynchronous context in which each module acts following an internal clock should be easy to implement without changing the strategy of the movement and without drastic modifications in the rules; in this case, though, the communication between the modules should be increased.

11.2 Open problems

There are different possible natural extensions for the treated problems; some of them have just been introduced due to lack of time, and some have not been treated in this thesis.

Overpassing of general shaped obstacles: the most interesting extension is the study of the overpassing of general shaped obstacles; a strategy for the locomotion in presence of a general connected obstacle is proposed in Chapter 10, but completing the systematic study of this problem has not been possible due to lack of time.

Locomotion of general shaped system: the existent rules for the shape transformation of 2-dimensional general-shaped system presented in [4] are easily adaptable to produce a locomotion of general shaped systems;

the rules by [4] produce the transformation of the shape of the system passing through an intermediate worm shape; the locomotion could then be achieved by applying our rules for the free locomotion of Chapter 2 to such intermediate shape, and blocking the advance of the system at the position in which we want to reconfigure the system into the goal configuration.

Notice that although the same existent rules could be applied to produce the locomotion of rectangular and histogram-shaped systems, the complexity of such rules is justifiable only in the case of the complex configurations for which they have been created, and not for regular systems for which a more simple set of rule that exploit their properties can be applied, as we have proved.

3-dimensional extensions: the locomotion of 3-dimensional rectangles and histograms could be achieved by an ideal subdivision of the system into different layers on the z-axis and the separate application of the same set of rules to each layer of the system, as proposed in [8] ; such extension is not immediate though, as the problems of disconnection of the layers need to be controlled with specific preconditions.

Hexagonal lattice systems: in the future, the sets of rules presented for rectangular modules could be extended to hexagonal lattice based systems. In this context, a few differences need to be considered in the adaptation of the rules; firstly, a grid position would have six neighbors, and each rule would need to be considered not in 4 but in 6 different directions, so an increment in the number of rules should be expected. Moreover, the study of the different types of bottlenecks should be modified, due to the differences in the shape of the modules.

References

[1] B.K. An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. Proc. of the *IEEE International Conference on Robotics and Automation* (ICRA), pages 3149-3155, 2008.

[2] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107-124, 2001.

[3] E. Yoshida, S. Kokaji, S. Murata, K. Tomita, and H. Kurokawa. Miniaturization of selfreconfigurable robotic system using shape memory alloy actuators. *Journal of Robotics and Mechatronics*, 12(2):96-102, 2000.

[4] F. Hurtado, E. Molina, S. Ramaswami, V. Sacristán. Distributed universal reconfiguration of 2D lattice based modular robots. *European Workshop on Computational Geometry*, Braunschweig (Germany), March 17-20, 2013.

[5] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: Mechanical design of a module for selfreconfigurable robotics. Proc. of the *IEEE International Conference on Robotics and Automation* (ICRA), pages 4095-4101, 2002.

[6] R. Wallner. *A system of autonomously self-reconfigurable agents*, Degree thesis, Institute for Software Technology, Graz University of Technology, Graz, Austria, 2009.

[7] Simulator: http://www-ma2.upc.edu/vera/AgentSystems/
Last visited: October 1srt, 2013.

[8] Z. Butler, K. Kotay, D. Rus, K. Tomita. Generic decentralized control for lattice-Based self-reconfigurable robots *The International Journal of Robotics Research* 2004; 23; 919