Distributed locomotion strategies for square lattice-based modular robots

Daniel Otero

Final degree thesis
Presentation date: 29th June 2016
Director: Vera Sacristán
Departament: Mathematics
22th June 2016

Bachelor Degree in Informatics Engineering
Mention in Computer Science
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Abstract

English

Previous work on locomotion strategies for lattice-based modular robots has provided several algorithms that allow modular robots move over sets of specific obstacles. Nevertheless, there was not yet a general algorithm that solved the problem of locomotion for any arbitrary set of obstacles. In this document, we introduce a general distributed algorithm of locomotion for 2D square lattice-based modular robots that solves the problem of locomotion over any given set of obstacles. The algorithm presented here is inspired in a caterpillar motion and uses the right-hand rule combined with local module information to guide the movement of a modular robot. The proof of correctness and complexity analysis are also given in this document alongside a complete functional implementation of the algorithm.

KEYWORDS - Lattice-based modular robots, distributed algorithms.

Spanish

Trabajos previos sobre estrategias de locomoción para robots modulares basados en celdas han proporcionado diversos algoritmos que permiten a este tipo de robots moverse en conjuntos específicos de obstáculos. Sin embargo, aún no existía un algoritmo general que solucionase el problema de locomoción para cualquier conjunto arbitrario de obstáculos. En este documento introducimos un algoritmo general de locomoción para robots modulares basados en celdas cuadradas 2D que soluciona el problema de locomoción sobre cualquien conjunto de obstáculos. El algoritmo presentado está inspirado en el movimiento de las orugas, también utiliza el principio de la mano derecha combinado con información local de cada módulo para guiar el movimiento de un robot modular. La demostración de corrección y el análisis de complejidad también son presentados en este documento junto una completa y funcional implementación del algoritmo.

KEYWORDS - Robots modulares basados en celdas, Algoritmos distribuidos.

Contents

Chapter 1

Introduction

This document contains all the information relative to a final project from a degree student of Computer Science at UPC. It consists on a distributed algorithm of locomotion for square lattice-based modular robots.

1.1 Context

Advances in engineering and physics of materials constantly improve the crafting of complex and sophisticated machines. Specially the field of Robotics benefits from them. It is well known that this field has experimented a substantial growth in terms of diversity and effectiveness on their devices. It is undeniable that the near future will be heavily influenced by the discoveries and projects that research teams are currently conducting in Robotics.

Computer Science and Mathematics are two areas that are strongly connected to the development of Robotics as they provide the essential tools and frameworks, as well as a set of algorithmic resources, that allow robots operate and perform their tasks.

One prominent and young branch of Robotics is *modular robots* (see Section 1.2). A lot of research is being done improving the design and abilities of these systems as well as their behavior and intelligence.

An important feature of modular robots is their behavior. Since they are a type of distributed system, they can be programmed with a distributed programming paradigm (see Section 1.2). Classical algorithms and protocols can work in this kind of machines, however, they do not perform as well as distributed algorithms.

This project focuses on a subgroup of modular robots called *Lattice-Based Modular Robots* (see Section 1.2). We will analyze most of the distributed locomotion strategies that have been created, try to improve them and even design new ones that contribute to the field of study. All the work and labor of this project will be targeted on the distributed algorithms and strategies that use lattice-based modular robots for their locomotion.

1.2 Relevant concepts

Here are defined key concepts related to the project.

Modular Robot [14] (also called *self-reconfigurable robot*) Autonomous kinematic devices which main feature is that they can greatly modify their shape. This is possible because of their morphology that is composed by several independent modules that can attach and detach from their neighboring modules, move relative to each other and, therefore, work together as an individual. Their modular body provides them a huge versatility, as opposed to the rigid-shaped single-purpose robots (see an example in Figure 1.1). All the system is considered asynchronous and modular. Consequently, a distributed programming paradigm suits them perfectly. Classical centralized algorithms also exists for modular robots, however, they introduce inefficiency into the system and reduce their scalability.

Distributed algorithms [13] Subject of Computer Science that studies the solution of problems related to systems composed of independent asynchronous processing entities with reduced memory, the ability of doing small computations and passing messages between them. Their procedures relay on the interaction between entities to reach optimal performance unlike classical algorithms where there is only one processing entity.

Lattice based modular robots [16] Type of modular robot that is characterized by the shape of its modules. Its modules are all shaped in a geometrical pattern that can tessellate the space; usually squares or hexagons in 2D and cubes in 3D. This property allows the system to interpret the environment as a grid. Their lattice architecture provides to these robots a huge scalability and simplicity to control (see an example in Figure 1.2).

Figure 1.1: Classical single-purpose robot (left) [1] vs modular robot (right) [3].

Figure 1.2: Lattice-based modular robot [2].

1.3 State of the art

1.3.1 Previous results

The robotics industry has created various models of lattice-based modular robots. This is a young field in its early phase. It has to evolve and it is currently under investigation. Nevertheless, there are already several models which have different abilities and can test the algorithms designed until now on real devices.

From a geometric point of view, there are different models of lattice-based modular robots regarding their movement patterns. All possible movements of modules are relative to their neighbors. The shape reconfiguration of the system is what generates the movement. In other words, a single module is not able to move by itself, it needs other connected modules to start moving relatively to them. Some can slide [5] (see Figure 1.3), some others rotate [11] (see Figure 1.4) and others can even squeeze into another module [15] (see Figure 1.5).

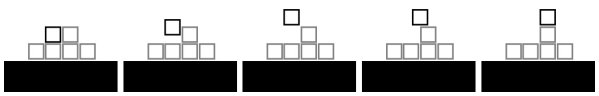

Figure 1.3: Sliding locomotion type of a square lattice-based modular robot.

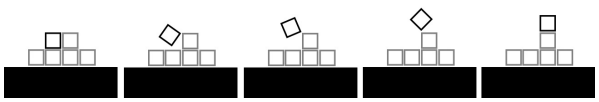

Figure 1.4: Rotating locomotion type of a square lattice-based modular robot.

Figure 1.5: Squeezing locomotion type of a square lattice-based modular robot.

A lot of work has been already done on the subject of locomotion. For 2D sliding squares, there are distributed algorithms that handle locomotion on free planes and also in the presence of some complex obstacles in an asynchronous way [12]. From another point of view, there are other proposals of locomotion in various synchronous models that also allow configurations of 2D sliding squares to move over complex obstacles and in a empty plane [6, 10]. There also exists approaches that implement well-known classical methods as path finding for 2D hexagons [17] and dynamic programming for 2D sliding squares and 3D sliding cubes [8].

1.3.2 Current issues

Despite having multiple approaches from different strategies, there is not a general distributed algorithm that can handle general complex obstacles. Nowadays there exists only specific strategies for certain obstacles and environment configurations [6, 12, 8]. This adds a handicap to the system that has to know in some way what it is going to face and then adapt to it. That situation makes lattice-based modular robots not really autonomous.

Also, there is a lack of a real asynchronous complete set of rules that can handle, individually, every kind of obstacle. Some work has been done on pseudo-asynchronous situations, using synchronous models of computation based on turns to emulate asynchronicity [6]. Regarding pure asynchronous models, there are solutions for simple obstacles but some complex configurations still cannot be overcome [12].

All current strategies rely on a certain morphology of the system, resulting on sets of rules that can only be applied effectively if the system is in a certain shape [11, 17]. General shapes would make those algorithms useless.

1.3.3 Challenges

Trying to generate a distributed algorithm for lattice-based modular robots comes with some challenges that every approach has to take into account. One of these threats is related to the integrity of the system; during all the process of locomotion, connectivity of all modules of the robot must be always maintained, meaning that the system does not split into separated connected components. Another challenge is to avoid collisions between modules, this means that not two or more modules try to go to the same position at the same time. Even another challenge is to avoid deadlocks, which

are situations where the morphological configuration blocks the movement of any module, so the system gets stuck and cannot move.

To ensure that those challenges are always overcome, current solutions rely on mathematical proofs of correctness. There are different approaches to this mathematical proofs. From one side there are proofs based in graph analysis [6]. Others in probabilistic analysis [8]. And finally some use exhaustive analysis to proof that under no condition the system will disconnect, collide or fall into a deadlock while moving [12].

1.4 Project description

1.4.1 Goal

The objective of this project is to provide efficient locomotion strategies for lattice-based modular robots. We will analyze the current state of locomotion strategies for this kind of robots and then try to improve the current ones or, if it shows to be possible, design a new and better set of rules. Moreover, we will aim to find a general distributed algorithm that can apply to any kind of obstacle.

After this first stages, the obtained results are to be implemented on a simulator to test their functionality.

1.4.2 Scope

This project has a vast horizon of possibilities and paths to take. There exist different lattice-based modular robots regarding the shape of their modules and the move requirements, as shown in Section 1.3.1. Those shapes are defined by any geometric patter that can tessellate the space.

To focus the efforts on a defined problem and work in an optimal manner, the project will study and work on 2D sliding square lattice-based modular robots. Other common types as hexagonal 2D or cubic 3D models are possible extensions that are not initially included in the scope of the project.

1.5 Structure of this document

The document is composed by nine chapters which can be divided into two different groups.

The first group forms the core of the text, which is the algorithm itself. This group chapters 2 to 5 and the Appendix. In this part of the document, the main ideas of the algorithm are explained. Also the proof of correctness and the complexity analysis of the algorithm are included. Moreover, there is an entire chapter devoted to explain the details of the algorithm alongside examples of implementation and illustrations. At the end of this document the whole algorithm implementation can be found. This set of chapters

explains incrementally the algorithm. We recommend to read all of them in order to completely understand the algorithm.

The second group is composed by the chapters 1, 6 and 7. This content explains the context, state of the art and statement of the problem that the algorithm solves. It describes the project management alongside the conclusions derived from the work and the references used.

Chapter 2

Algorithm description

The proposed algorithm solves the general problem of locomotion for a square lattice-based modular robot. Since it is intended to work in a distributed system, it follows the distributed paradigm of computation. The algorithm is composed by a set of rules which are constantly evaluated by every module of the system and applied if it is possible.

All rules use only local information and do not use message passing. The algorithm works with the model of sliding motion and ensures that no collision, disconnection or deadlock occurs during the locomotion of the robot for any arbitrary set of obstacles.

2.1 Definitions

In order to describe the algorithm and its features, we will define here some terms and concepts that will be used in the following explanations.

Active module Module that is moving.

Static module Module that is not moving.

Movable module Module that can be moved without disconnecting the system following the right-hand rule.

Base Set of static modules upon where active modules move. Geometrically, they form a path composed by not movable static modules.

Leaf module Movable module that is not part of the base and is only connected to another module.

Pivot Movable static module that is not part of the base and also not a leaf.

Head The most advanced static module of the base of the robot, in terms of the distance from the module to the destination following the perimeter of the obstacle.

Tail The rearmost static module of the base of the robot in terms of the distance from the module to the destination following the perimeter of the obstacle.

Wall deadlock Situation induced by a narrow obstacle where none of the leaves of the system can move because of the morphology of the obstacle. Our strategy consists in moving a pivot (illustrated in Figure 2.1 in yellow) in order to move again the robot.

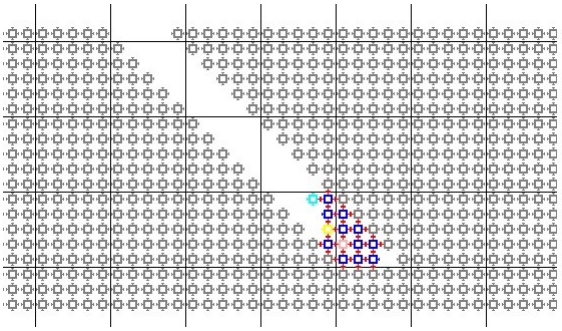

Figure 2.1: Wall deadlock.

Perfect cycle Configuration of the system where the head is stuck in a cycle and there are no more modules to move. See Figure 2.2 for an illustration.

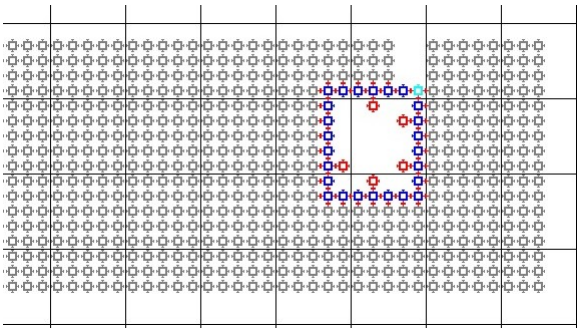

Figure 2.2: Perfect cycle.

Tailed cycle Configuration of the system where the robot has a cycle and the head is outside it. Visually the cycle seems to have a tail. See Figure 2.3 for an example.

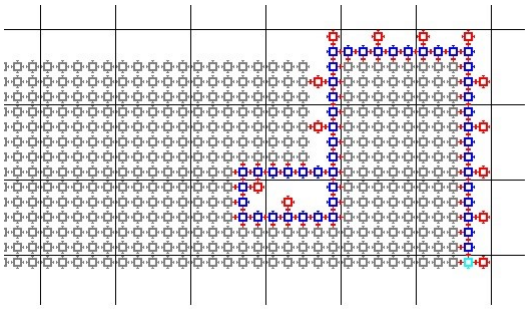

Figure 2.3: Tailed cycle.

T structure Distinctive robot shape pattern that only appears in tailed cycles. It is used to identify the module that will break the cycle and deduce to which direction move. See Figure 2.4 for an illustration.

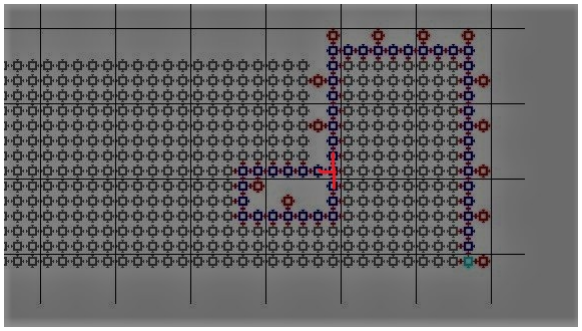

Figure 2.4: T structure marked on red.

2.2 Locomotion

The algorithm induces a caterpillar movement following the right hand rule [9] in order to guide the robots in their east locomotion. The algorithm basically distributes the modules into a two level configuration with the base in one level and in the other, all active modules which are moving upon the base. When an active module reaches the head of the system, it positions itself in the next advanced position of the base and becomes the new head. Once a static module realizes that it is a leaf and can move following the right hand rule, it activates and moves towards the head of the system upon the base. This previously mentioned situation can only occur to the the tail module of the base. See an example of this principle of locomotion in Figure 2.5.

Following this principle, the robot moves following the perimeter of any obstacle as long as the obstacle bottlenecks are wide enough to let the robot

pass; which is a width that corresponds to four times the length of a module (two lattice positions to locate the base of static modules, plus another two to be used by the moving active modules).

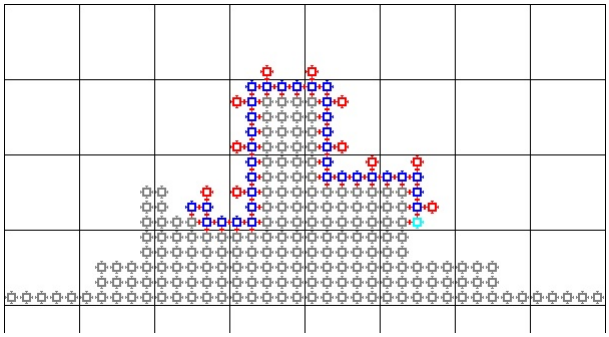

Figure 2.5: Caterpillar locomotion over obstacles.

2.3 Narrow dead ends

The strategy explained in the previous section works for almost all obstacle configurations. However, there are some obstacles whose morphology impedes that with only the prior principle the system arrives to its destination. The path of the robots can have bottlenecks that the robot must enter and exit in order to continue its locomotion. If bottlenecks are narrow enough, the robot will not be able to maintain the caterpillar strategy without producing deadlocks and collisions. Those special obstacle configurations are bottlenecks of narrowness strictly smaller than four lattice positions.

In those situations, our algorithm will fill the space from the end of the bottleneck to its exit. Pivot modules will appear inside the filled bottleneck. When the bottleneck is filled, the robot will continue moving active modules upon the base, as it will consider the modules that close the bottleneck as belonging to the base. Once all modules have passed upon the bottleneck, it will be emptied from the top to the tip following the right-hand rule. Modules will activate as always, however, pivot modules have higher priority of activation than basic static modules, inducing a right order of emptying the bottleneck, i.e., avoiding collisions in deadlocks. (The proof of correctness can be found in Section 3.2.2). See an illustration in Figure 2.6.

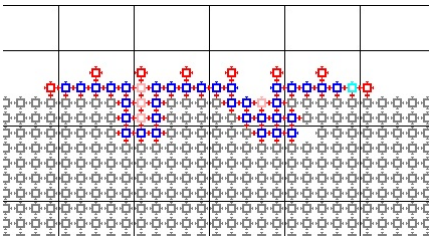

Figure 2.6: Narrow bottlenecks in different directions

Eventually, the system may fall in a configuration where the no leaf can move, a wall deadlock. In these cases, a pivot module will activate and move, triggering the system to move again. See Figure 2.1.

2.4 Cycles

Since obstacles can have any form, it may happen that a narrow bottleneck is followed by a large space where the system can move in a caterpillar formation. If this happens near a dead end and the robot is long enough, a cycle of modules around an empty space may get formed. In this cycle, active modules may get confined inside it in a deadlock configuration. These cycles can be found in two different configurations: tailed cycles (see Figure 2.3) or perfect cycles (see Figure 2.2). In order to solve this problem, we will identify a pivot which will be crucial to break the cycle without disconnecting the robot and allow the system to move correctly, by locating the T structure (see Figure 2.4).

Like what happens in bottlenecks, cycles will be opened once all remaining movable modules outside the cycle have passed upon it.

Collisions between active modules moving over the head and other active modules advancing in opposite directions may happen since a cycle induces that at the time of closing the cycle, the head will be neighboring a static module different from its follower. Checking each time that an active module moves does not have in opposite direction another active moving over a head is enough to prevent the collisions. If one collision is predicted, one of the involved active modules will stop in order to to avoid the collision and continue the locomotion.

In every exit of a tailed cycle, there will be a T-shaped formation. Analyzing the T-shaped formation, one can find a pivot at the exit of any cycle and also deduce to which direction it has to move. Finding which direction to move the T pivot is crucial to dodge infinite looping movements of pivots in a cycle (please find the proof of correctness of this strategy, Section 3.2.3). Using this information we can move through obstacles with cycles with no fear of getting stuck or disconnected and move to the correct direction. The trick is to realize that in one side the T structure will have the gap of the

cycle and in the other side the piece of obstacle that closed the cycle.

Exceptionally, if the space of the cycle and the number of modules of the system is the same, the robot will get stuck in a perfect cycle formation where there is no T structure to start the disassembling. However, it is quite easy to identify which module is the pivot and which direction to move it: when the head meets the tail, the tail becomes a pivot that has to break this specific cycle configuration moving upon the head.

Chapter 3

Proof of correctness

The proof of correctness must show that the algorithm always works in any scenario moving the robot from the starting point towards its east without ever disconnecting, colliding and falling into deadlocks.

3.1 Hypothesis

Let us define the offset of the obstacles (and the ground) as the set of all lattice positions which are adjacent to an obstacle (or the ground). See Figure 3.1 for an illustration.

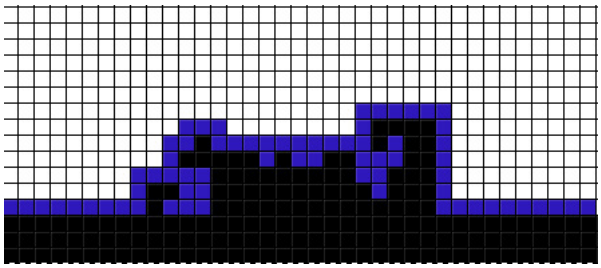

Figure 3.1: Floor and obstacles in black, offset positions in blue.

These are the positions that may eventually be occupied by the static modules as the robot advances.

Consider now the sequence of lattice cells that an active module may occupy when traversing the scene from west to east. All these cells are adjacent to an offset cell. Connect their centers in the order defined by the right-hand rule. The result is a directed graph. One can also think of this digraph as a path, by duplicating its bi-directional edges. If preferred, one can also only refer to the underlying undirected graph. See an example in Figure 3.2.

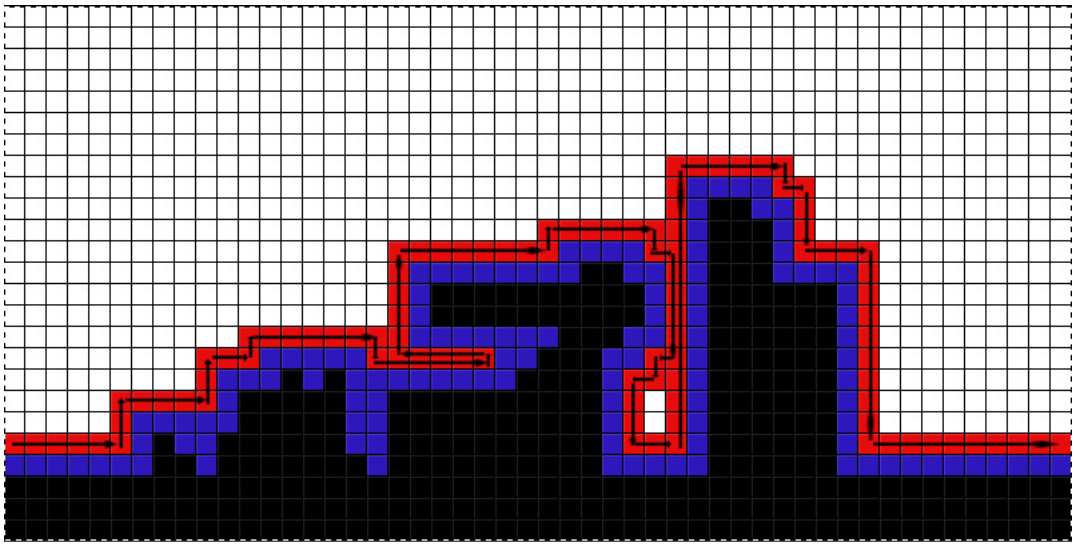

Figure 3.2: Floor and obstacles in black, offset cells in blue, and active positions in red with the underlying directed graph.

Three situations may happen:

1. The digraph is a simple path. In other words, the underlying graph is a path.

2. The path has self-intersections, but the non-intersecting edges of it form one connected component. In other words, the underlying graph is a tree.

3. The path has self-intersections, and the non-intersecting edges form several connected components. In other words, the underlying graph contains cycles.

3.2 Proof

We will now prove that our strategy works in any of the three cases.

3.2.1 Case 1: The underlying graph is a path.

Our strategy is to move active modules in the path following the right-hand rule.

Therefore:

- No collisions or deadlocks may happen since active modules move in a path.

- The right hand rule is successful in moving the robot from the start to its destination since there is a path that connects them.
- Connectivity if guaranteed by the fact that only leaves activate and move.

3.2.2 Case 2: The underlying graph is a tree.

Our solution combines the right hand rule strategy together with a 'filling the gap' strategy which fills the branches from their tip to their branching point and finally empties them from their branching point to the tip once all the remaining modules have passed it, as shown in Figure 3.3.

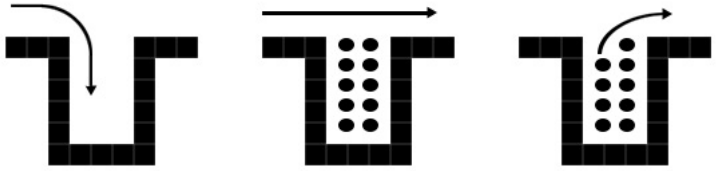

Figure 3.3: Steps of the 'fill the gap' strategy.

Therefore:

- No collisions happen since all active modules or are following the path or filling or emptying a branch. Since the filling process of a branch is done from the tip to the branching point and the emptying process inversely, following the right-hand rule, no collision can occur.
- By definition, branches are filled by static modules which are not part of the base and can be moved without disconnecting the system, the so called pivots. Since all branches have pivots, the robot will always have a module to move. For this reason, the movement cannot fall into a deadlock in branches. Also, as proven before, in a path the system cannot fall into a deadlock.
- The remaining of the previous proof applies.
- In order to finish the proof of this case, only a fact remains to be proven, namely that all branches are detected. The detection of branches is done while the algorithm is running. A branch is detected when a module deactivates neighboring a head but does not make the head

advance. This module will be a pivot. It can be seen that this situation can never happen on a path, since all active modules can only deactivate in the position in front of the head. It can also be seen that in the tip of a branch this occurs, since the active module that arrives at the end of the branch gets stuck between the base/offset modules, and the head that has arrived to the end of the branch and now will climb out of it. The branch will continue growing 'filling the gap' extending the tip of the branch until the head continues advancing in the path. With this method it can be seen that all branches will be detected.

3.2.3 Case 3: The underlying graph contains cycles.

Our strategy is to locally recognize the existence of a cycle and to open it up when all the remaining modules exterior to the cycle have passed. Figure 3.4 show the main steps of this strategy.

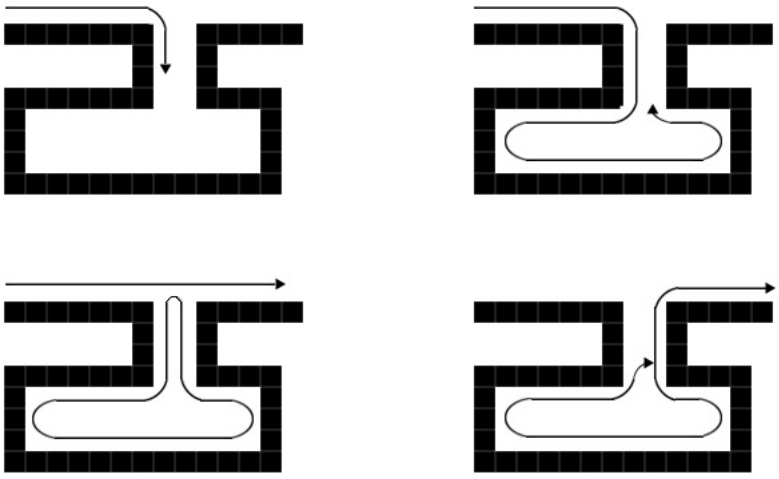

Figure 3.4: Mian stages of our strategy for cycles.

Therefore:

- No collisions happen since all active modules will follow their path until the closing of the cycle where, just looking one step ahead, active modules can dodge collisions. In order not to stop the two active modules that would collide and deadlock the robot, one of them waits still while the other one moves and closes the cycle. This module will

be the one neighboring the head. It can be seen that since the head is always at the front of the robot, the head will always be involved in the closing of a cycle while the robot is moving. Then the head will continue if more modules pass upon the cycle. Checking that no active module is passing upon the cycle ensures that there will be no collision when opening the cycle. Since the opener will move following the right-hand rule and will end in a path, no further collision can happen.

- Since all cycles are detected and eventually opened, the robot will never fall into a deadlock. Proving that all cycles can be detected is simple. There are two types of cycles: perfect cycles and tailed cycles. The perfect cycles have, by definition, the head neighboring the tail. Tailed cycles have, by definition, a T structure where the cycle meets the path. So there are local characteristic features that allow to identify all cycles and the pivots to move open.

- The remaining of the previous proof applies.

- In order to finish the proof of this case, only one fact remains to be proven, namely that the cycle breaker moves in the correct direction. It can be seen that the cycle breaker module has, following the right-hand rule, two options to move: to enter again the cycle or to go exit it. Eventually, entering again the cycle may lead to a kind of dynamic trap situation where the cycle breaker enters always the cycle and the robot has not enough time to disassemble the cycle when an inner cycle module closes again the cycle. If this repeats infinitely, the robot will be stuck. For this reason, moving always the cycle breaker towards the exterior of the cycle is mandatory. I can be easily seen that a cycle has always an empty space in the direction of going inside the cycle, and always an obstacle piece in the opposite direction, which previously induced the active paths to close. So moving always in the direction of the obstacle and not the gap ensures the correct choice.

Chapter 4

Complexity

The computational complexity of the algorithm can be analyzed from various perspectives. However, any classical conception of complexity has to be left behind since the algorithm follows the distributed paradigm of computation.

4.1 Number of parallel steps

The complexity of the algorithm can be analyzed in terms of the number of steps that will take a robot of n modules to reach an arbitrary target position in an hypothetical synchronous execution.

To analyze this computational cost, we use a notion of distance between the robot modules and their destination. Since the algorithm interprets the space as a grid, we can use the number of cells that form the path the robot modules will follow to reach their goal. As mentioned in the description of the algorithm, the robot moves over the perimeter of the obstacle until reaching the destination. Let us consider this distance to be m lattice cells.

We can state that the robot has reduced the distance m between the starting point and the goal to $m - 1$ when the head the system has moved one position closer to it.

In obstacles without narrow caves and cycles, it can be seen that, since the algorithm moves the active modules over the base and activates the tail when there is not any active upon it, the robot will always have a train of active modules moving upon the base with distance k between them, being that k a small constant number. Having that in mind we can say that if the modules start moving already in a caterpillar formation, the system will take k parallel steps to advance the head one position. This makes the complexity linear, in an order of $k*m$, what means $O(m)$. In the worst case, the robot starts as a line, the complexity is linear to m plus the distance that the first module to activate has to move until reaching the head, which is n, the number of modules of the robot. The following modules will create the aforementioned chain of movements. In this case, the cost of moving the

first active module is in the order of n leading to a complexity of $O(m+n)$.

In narrow dead ends we can see that they are filled and emptied following the right hand rule without interfering or delaying the train of moving active modules. Therefore, the computational cost is $O(m+n)$.

In cycles we can see that the chain of movement of active modules break while identifying the pivot that will break the cycle, having to start again the rhythm of movement of the robot. This process is similar to the one faced having as starting morphology a line, which takes a cost proportional to n. Since any path can have an arbitrary number of cycles, but their lenghtd will always add up to a number smaller than n, this situation will lead to the same asymptotic cost $O(m+n)$.

4.2 Number of moves per module

Since all movements follow the right hand rule, any movement will bring closer the module to the destination which is at distance m, leading to an initial cost of $O(m)$, as long as there are no cycles.

Indeed, in narrow dead ends the modules also fill and empty the caves following the right hand rule, which ensures that every movement reduces the distance to the destination.

Only an active module trapped in a cycle can perform extra steps that do not reduce its distance to the destination. The number of extra steps that an active module can do in a cycle before the cycle is dismantled and the module advances again, the number of modules not trapped in the cycle, which is smaller than n. That is because, in the worst case scenario, the detection and movement of a cycle breaker will take the time of passing all the external modules of the cycle upon it, leaving the cycle breaker movable. The complexity in this scenarios is $O(m+n)$.

In the worst case, a module may get consecutively trapped in all cycles. But for this to happen, the distance between consecutive cycles must close to n, since this is the only way for a module to get trapped again, as it needs first to walk upon the entire base. Therefore, the total number of cycles can only be roughly m/n. Therefore such module would perform $O((m/n) * n) = O(m)$ extra moves, which will not increment the overall complexity regarding the number of moves per module.

Chapter 5

Detailed description of the implementation

In this section of the document the details of the algorithm will be presented. The explanations will be organized around the rules and concepts behind them. Each complex idea will be followed by an example of the implementation that we've done and images to represent the local scenario. All the implementations are coded with the language of the Simulator [7] used to develop the project.

The whole implementation can be found at the end of this document in the Annex 1.

5.1 Implementation language

As said before, the implementation of the algorithm was made with the language of a simulator given by the project adviser. That simulator has its own language for coding. That language follows the declarative paradigm of programming generating programs that are sets of rules. Each rule has its name, priority, precondition and action. An action will be done at a given time if the precondition is fulfilled and there is no another applicable rule with higher priority at the same time.

5.1.1 Language syntax

This are the instructions used in the following examples of implementation. We will mention them in order to provide the reader the tools to understand the chunks of code. More information available at the user's guide of the Simulator [7].

S—— Declaration of a new state / checking if module has the given state / Change state to given state.

E-,- Check if empty space.

T-,-,—— Check if the there is a module with a specific state at a given position (local to the processing module).

A—- Check if attached modules in the given directions / Attach to the modules in the given direction.

P-,- Move to a given position (local to the processing module).

A consideration worth mentioning is that the language constructs the sentences as conjunctions of clauses. That means that there is not the logic operation or implemented and it needed, a logically equivalent sentence to an or has to be build using conjunctions of clauses.

5.2 States

Our implementation interprets the space as a 2D squared grid where each cell has a state. This are the following states that the set of rules use to interpret the space:

5.2.1 Environment

Obstacle Piece of an obstacle from the environment.

Empty Empty space.

5.2.2 Modules of the robotic system

Stopped Module that is not moving, waiting to be able to change state. See an example in Figure 5.1.

Active Module that is moving following the right-hand rule of movement. See an example in Figure 5.1.

Head Stopped module which position corresponds to the most advanced module of the system regarding the path that the system will follow in order to arrive to its destination. Just one per system. See example in Figure 5.1

Pivot Stopped non leaf module that theoretically can be moved without the risk of disconnecting the system, however, no rule of movement can be applied to them. See example in Figure 5.1.

Superactive Module that moves using rules that not check if the movement disconnects the system. Only a pivot can become superactive. They are used to break wall deadlocks. See example in Figure 5.1.

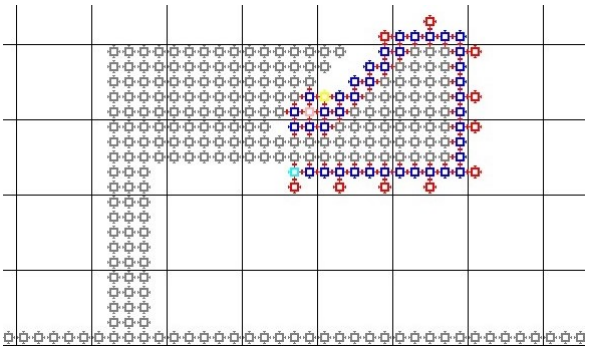

Figure 5.1: Stopped (dark blue), actives (red), head (light blue) pivot (pink) and superactive (yellow)

Perfect Module that appears in perfect cycles just near a head. It is used to break perfect cycles deadlocks. See example in Figure 5.2

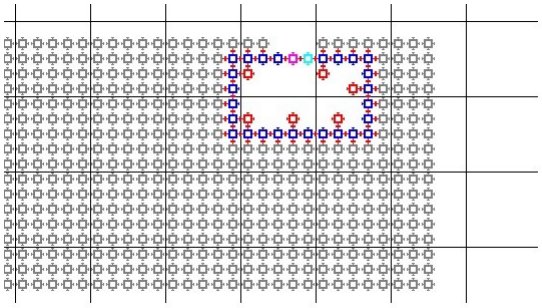

Figure 5.2: Perfect module (magenta)

N / S / E / W / NE / NW / SE / SW cycle breach used to break tailed cycle deadlocks. Depending on the direction on which they have to move to correctly break the cycle they adopt one tag or another, however, the idea behind all of them is the same. See example in Figure 5.3

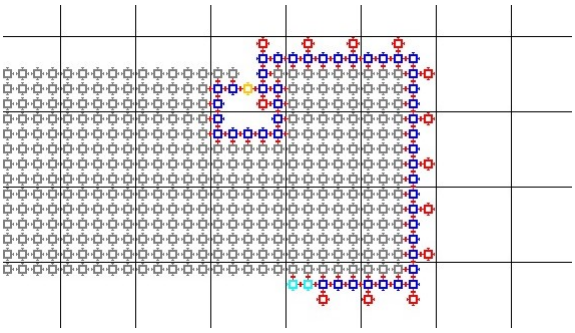

Figure 5.3: T module module (magenta)

5.3 Rules

All this rules individually solve one aspect of the problem of locomotion for square lattice-based modular robots. They are thought to be used in a distributed system in conditions of either synchronicity or asynchronicity. Working together, they compose a general algorithm of locomotion.

Conceptually there are 18 rules but depending on the implementation one can have more or less real rules. In this case, the implementation consists on 50 rules.

5.3.1 Movement by active

2 conceptual rules implemented in 8 different rules
This rules can only be applied to active modules. They implement the movement following the right hand rule, which can be done in the eight cardinal directions if the active module is movable and has a base of static modules to move upon. Also there are checks for ensuring to not collide with other modules. We can distinguish two type of rules depending if they move in diagonals or straight directions.

Straight direction movement rule explained more in depth using as example the 'N movement' rule:

Precondition:

- Check if active: `SACTIV` - Check if movement is possible (see Figure 5.4): `E0,1 !(!E-1,0 !T-1,0,OBSTA !T-1,0,ACTIV) !(!T1,1,STOPA !T1,1,PIVOT !T1,1,HEADD !T1,1,SUPER !T1,1,PERFE !T1,1,SECYC !T1,1,NWCYC !T1,1,SWCYC !T1,1,NECYC !T1,1,WCYCL !T1,1,ECYCL !T1,1,SCYCL !T1,1,NCYCL) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,SUPER !T1,0,PERFE !T1,0,SECYC !T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,NCYCL)`

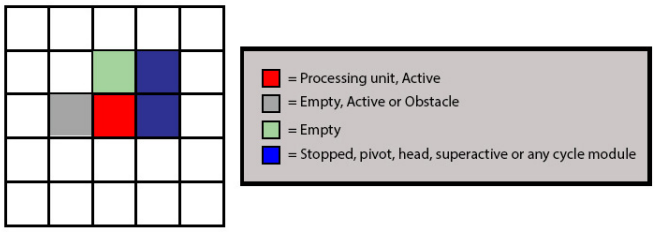

Figure 5.4: Check if movement is possible, N movement

- Check that there is not the head in front of you and in opposed direction with an active module on it that is going to collide with you (see Figure 5.5): `!(T-1,0,OBSTA E0,2 T-1,2,HEADD T0,3,ACTIV !(!T-1,3,STOPA !T-1,3,PIVOT))`

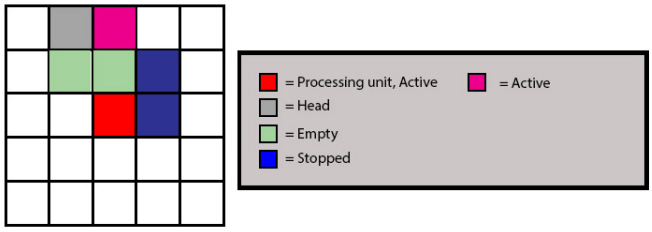

Figure 5.5: Prevent direct collision, N movement

- Check if there is the head in front of you and in opposed direction with an active modules going to be over it creating a configuration where an empty space in front of the head is going to appear at the exit of a cycle and so stuck the head on that position, if you just wait the opposite module will fill that gap in front of the head and the head will move correctly (see Figure 5.6): `!(T-1,0,OBSTA E0,2 T-1,2,HEADD T0,3,ACTIV !(!T-1,3,STOPA !T-1,3,PIVOT))`

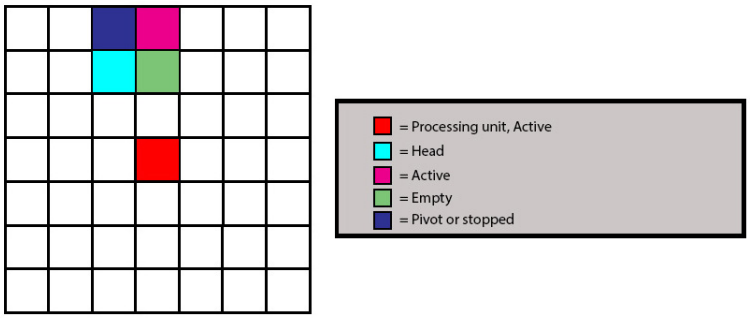

Figure 5.6: Prevent degenerated configuration, N movement

Action:

- Move north and attach to all possible modules

Diagonal direction movement rule explained more in depth using as example the 'NE movement' rule:

Rule explained more in depth

Precondition:

- Check if active: `SACTIV`

- Check if movement is possible (see Figure 5.7): `E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA !T-1,0,ACTIV) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,SUPER !T1,0,PERFE !T1,0,SECYC !T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,NCYCL)`

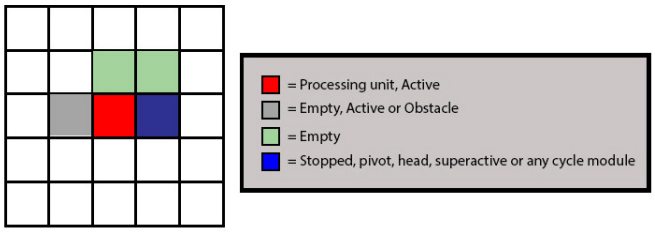

Figure 5.7: Check if movement is possible, NE movement

- Check if there is the head in front of you and in opposed direction with an active module on it that is going to collide with you (see Figure 5.8): `!(T1,0,STOPA T0,2,HEADD T1,2,ACTIV E1,1)`

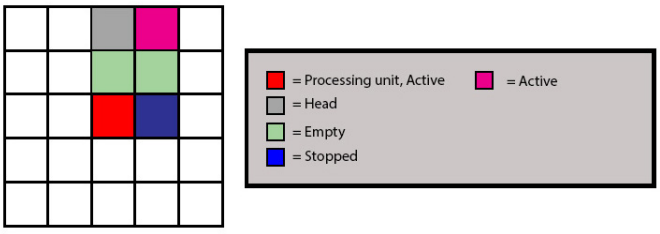

Figure 5.8: Prevent collision, NE movement

- Check if there is the head in front of you and in opposed direction with an active module going to generate a collision because of the creation of a cycle (see Figure 5.9): `!(T0,2,HEADD E2,1 E1,2 T2,2,ACTIV !(!T1,3,STOPA !T1,3,PIVOT))`

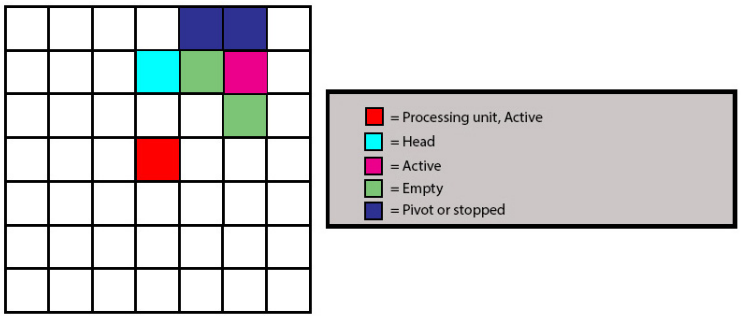

Figure 5.9: Prevent degenerated configuration, NE movement

Action:
- Move northeast and attach to all possible modules

5.3.2 Deactivation of active

1 rule
When active has no possible movements to perform and rules to apply to itself it becomes stopped.

5.3.3 Activation of stopped

1 conceptual rule, split in 2 to not generate complex code
When a stopped module becomes leaf, has no active or superactive modules on its directly near 8 direction neighbors and can apply a movement rule, it becomes active. Moreover, there are extra conditions to prevent the system disconnect.

Straight direction activation rule of stopped modules explained more in depth, it corresponds to the 'Activation by stopped N S E W' rule:

Precondition:
- Check if stopped: `SSTOPA`
- Check if movement is possible in N S E W directions:

```
!(!(E0,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,1,STOPA !T1,1,PIVOT
!T1,1,HEADD !T1,1,PERFE !T1,1,SECYC !T1,1,NWCYC !T1,1,SWCYC
!T1,1,NECYC !T1,1,WCYCL !T1,1,ECYCL !T1,1,SCYCL !T1,1,NCYCL)
!(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,SECYC
!T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,ECYCL
!T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1 !(!E1,0 !T1,0,OBSTA)
!(!T-1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,HEADD !T-1,-1,PERFE
!T-1,-1,SECYC !T-1,-1,NWCYC !T-1,-1,SWCYC !T-1,-1,NECYC
!T-1,-1,WCYCL !T-1,-1,ECYCL !T-1,-1,SCYCL !T-1,-1,NCYCL)
!(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD !T-1,0,PERFE
```

!T-1,0,SECYC !T-1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC
!T-1,0,WCYCL !T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL))
!(E1,0 !(!E0,1 !T0,1,OBSTA) !(!T1,-1,STOPA !T1,-1,PIVOT
!T1,-1,HEADD !T1,-1,PERFE !T1,-1,SECYC !T1,-1,NWCYC
!T1,-1,SWCYC !T1,-1,NECYC !T1,-1,WCYCL !T1,-1,ECYCL
!T1,-1,SCYCL !T1,-1,NCYCL) !(!T0,-1,STOPA !T0,-1,PIVOT
!T0,-1,HEADD !T0,-1,PERFE !T0,-1,SECYC !T0,-1,NWCYC
!T0,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !T0,-1,ECYCL
!T0,-1,SCYCL !T0,-1,NCYCL)) !(E-1,0 !(!E0,-1 !T0,-1,OBSTA)
!(!T-1,1,STOPA !T-1,1,PIVOT !T-1,1,HEADD !T-1,1,PERFE
!T-1,1,SECYC !T-1,1,NWCYC !T-1,1,SWCYC !T-1,1,NECYC
!T-1,1,WCYCL !T-1,1,ECYCL !T-1,1,SCYCL !T-1,1,NCYCL)
!(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC
!T0,1,NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,ECYCL
!T0,1,SCYCL !T0,1,NCYCL)))

- Check if no actives on its 8 direct neighbors: !T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV

- Check if no superactives on its 8 direct neighbors: !T-1,1,SUPER !T0,1,SUPER !T1,1,SUPER !T1,0,SUPER !T1,-1,SUPER !T0,-1,SUPER !T-1,-1,SUPER !T-1,0,SUPER

- Check for not being a Z-corner (four possible directions) (see Figure 5.10): !(!(!T-1,0,STOPA !T-1,0,PIVOT) E1,0 !(!T0,-1,STOPA !T0,-1,PIVOT) !(!T1,-1,STOPA !T1,-1,PIVOT !T1,-1,HEADD) !(!T-1,-1,OBSTA !E-1,-1)) !(!(!T1,0,STOPA !T1,0,PIVOT) E-1,0 !(!T0,1,STOPA !T0,1,PIVOT) !(!T-1,1,STOPA !T-1,1,PIVOT !T-1,1,HEADD) !(!T1,1,OBSTA !E1,1)) !(!(!T0,1,STOPA !T0,1,PIVOT) E0,-1 !(!T-1,0,STOPA !T-1,0,PIVOT) !(!T-1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,HEADD) !(!T-1,1,OBSTA !E-1,1)) !(!(!T0,-1,STOPA !T0,-1,PIVOT) E0,1 !(!T1,0,STOPA !T1,0,PIVOT) !(!T1,1,STOPA !T1,1,PIVOT !T1,1,HEADD) !(!T1,-1,OBSTA !E1,-1))

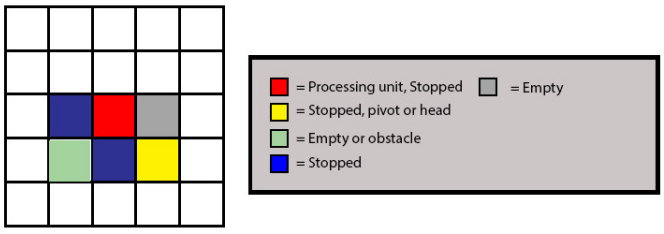

Figure 5.10: Check if not being a Z shaped corner

Action:

- Module state changed to active

Diagonal direction activation rule of stopped modules explained more in depth, it corresponds to the 'Activation by stopped NE SW NW SE' rule:

Precondition

- Check if stopped: `SSTOPA`

- Check if movement is possible in NE SW NW SE directions: `!(!(E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,SECYC !T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1 E-1,-1 !(!E1,0 !T1,0,OBSTA) !(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD !T-1,0,PERFE !T-1,0,SECYC !T-1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL !T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL)) !(E-1,0 E-1,1 !(!E0,-1 !T0,-1,OBSTA) !(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC !T0,1,NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,ECYCL !T0,1,SCYCL !T0,1,NCYCL)) !(E1,0 E1,-1 !(!E0,1 !T0,1,OBSTA) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD !T0,-1,PERFE !T0,-1,SECYC !T0,-1,NWCYC !T0,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !T0,-1,ECYCL !T0,-1,SCYCL !T0,-1,NCYCL)))`

- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`

- Check for not being a corner (four possible directions) (see Figure 5.11): `!(!(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD) !(!T1,1,OBSTA !E1,1) !(!T0,-1,OBSTA !E0,-1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD) !(!T-1,1,OBSTA !E-1,1) !(!T0,-1,OBSTA !E0,-1) !(!T1,0,OBSTA !E1,0)) !(!(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!T1,-1,OBSTA !E1,-1) !(!T0,1,OBSTA !E0,1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!T-1,-1,OBSTA !E-1,-1) !(!T0,1,OBSTA !E0,1) !(!T1,0,OBSTA !E1,0))`

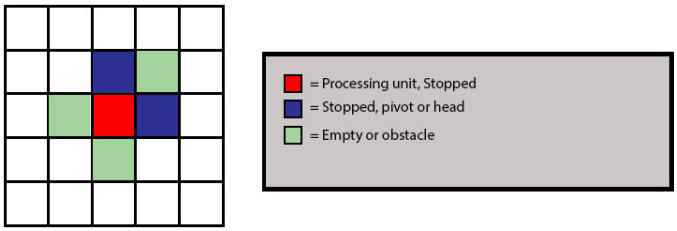

Figure 5.11: Check if not being a coner

- Check if not only a pivot will be the base of movement (right order of disassemble, pivots have higher priority) (see Figure 5.12): `!(!(!E-2,0 !T-2,0,ACTIV) T-1,0,PIVOT) !(!(!E2,0 !T2,0,ACTIV) T1,0,PIVOT) !(!(!E0,2 !T0,2,ACTIV) T0,1,PIVOT) !(!(!E0,-2 !T0,-2,ACTIV) T0,-1,PIVOT)`

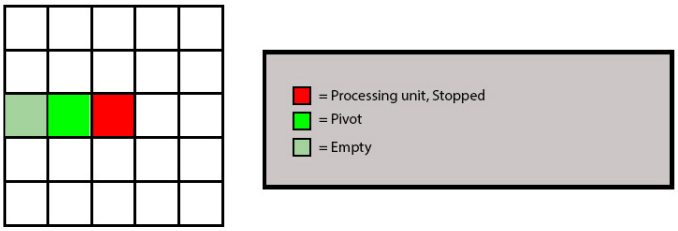

Figure 5.12: Check that you follow the right order of activation

- Check if your are the last module of a broken cycle to wait to the right moment to activate in order to disassemble the cycle correctly that is when no active module is going towards you, in the gap of inner movers of the cycle (see Figure 5.13): `!(E0,1 E1,0 T1,1,STOPA !(!T2,1,STOPA !T2,1,PIVOT) T2,0,ACTIV) !(E0,-1 E-1,0 T-1,-1,STOPA !(!T-2,-1,STOPA !T-2,-1,PIVOT) T-2,0,ACTIV) !(E1,0 E0,-1 T1,-1,STOPA !(!T1,-2,STOPA !T1,-2,PIVOT) T0,-2,ACTIV) !(E-1,0 E0,1 T-1,1,STOPA !(!T-1,2,STOPA !T-1,2,PIVOT) T0,2,ACTIV)`

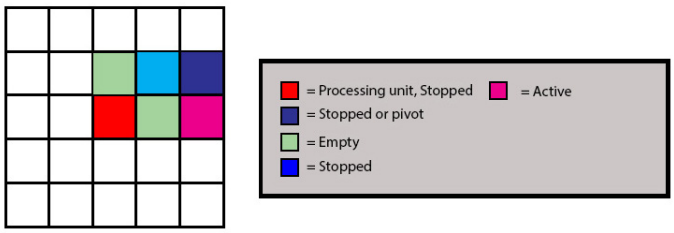

Figure 5.13: Check if right moment to active in a cycle

Action:

- Module changed to active.

5.3.4 Pivot detection

1 rule

When an active or stopped module is positioned between an inactive module and a head, and the head is positioned over another inactive module (this can be deduced analyzing the configuration having in mind that the movement follows the right hand rule). The module becomes a pivot (see Figure 5.14).

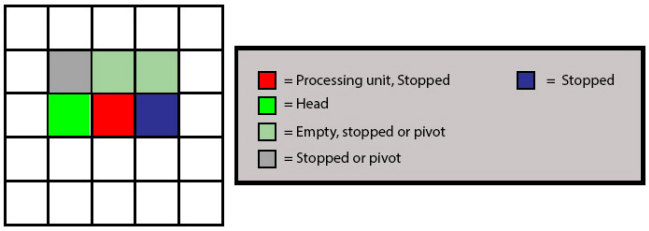

Figure 5.14: Pivot detection

5.3.5 Extra pivot detection oriented to superpivot

1 rule

There are some obstacle formations that generate accumulations of modules when the head has long passed, so they are undetected pivots; for those moments, if we find configurations where a module has properties of pivot and also of possible superpivot, it is tagged as pivot (see Figure 5.15).

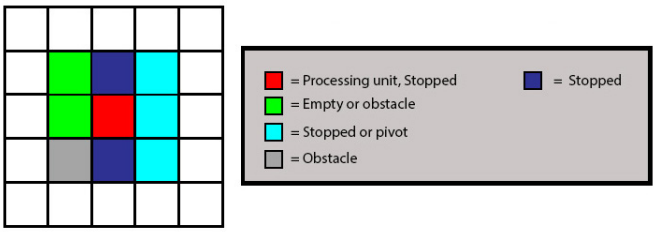

Figure 5.15: Pivot detection looking for superpivot

5.3.6 Activation of pivot

1 conceptual rule, split in 2 in order to not generate complex code
When a pivot module becomes leaf, has no active or superactive modules on its directly near 8 direction neighbors and can apply a standard movement rule, it becomes active. Moreover, there are extra conditions to prevent the system disconnect.

Straight direction activation rule of pivots explained more in depth, it corresponds to the 'Activation by pivot N S E W' rule:

This rule is exactly the same as the straight movement activation from stopped but applied to pivots.

Diagonal direction activation rule of pivots explained more in depth, it corresponds to the 'Activation by pivot NE SW NW SE' rule:

This rule follows the same principles as the activation of stopped because of diagonal movement but with the exception that this rule does not have on its precondition the rule that gives priority of disassemble to pivots.

5.3.7 Pivot deactivation

1 rule
When a pivot module looses its features that characterize it of pivot, it becomes stopped.

Precondition:

- Check if pivot: `SSTOPA`

- Check if the module is a corner of the system; empty corners cannot be pivots because their loose the propoerty of being able to be moved without disconnecting the system (see Figure 5.16):

```
!(!(!T1,0,STOPA !T1,0,HEADD !T1,0,PIVOT) E-1,0 (!T0,-1,STOPA
!T0,-1,HEADD !T0,-1,PIVOT) E0,1 E1,-1) !(!(!T-1,0,STOPA
!T-1,0,HEADD !T-1,0,PIVOT) E1,0 !(!T0,1,STOPA !T0,1,HEADD
!T0,1,PIVOT) E0,-1 E-1,1) !(!(!T1,0,STOPA !T1,0,HEADD
!T1,0,PIVOT) E-1,0 !(!T0,1,STOPA !T0,1,HEADD !T0,1,PIVOT)
```

E0,-1 E1,1) !(!(!T-1,0,STOPA !T-1,0,HEADD !T-1,0,PIVOT) E1,0
!(!T0,-1,STOPA !T0,-1,HEADD !T0,-1,PIVOT) E0,1 E-1,-1)

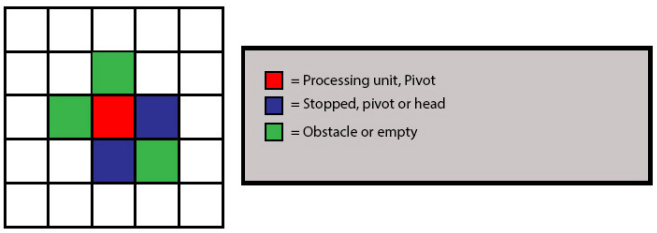

Figure 5.16: Pivot deactivation

5.3.8 Head transmission

1 rule
When an active module is about to become stopped having a head as neighbor of connectivity 8, an obstacle as neighbor with connectivity 4 and with not a head in its diagonals (when head is at the exit of cycles, it could go 'back' to the tail if not), that means that it now has become the head. In further rules the old head will become stopped.

5.3.9 Head deactivation

1 rule
When a head module has another head looses its features of head becomes a stopped module.

Precondition:
-Check if the current state is head: SHEADD
- Check if the current module has a head as neighbor and, considering the other head neighbor as the local north, an obstacle to its local east or northeast upper right. The module is not longer the head of the system because the right hand motion that follows the system induces that the other head is in a more advanced position than the current module (see Figure 5.17): !(T0,1,HEADD T1,0,OBSTA) !(T-1,0,HEADD T0,1,OBSTA) !(T0,-1,HEADD T-1,0,OBSTA) !(T1,0,HEADD T0,-1,OBSTA) !(T0,1,HEADD T1,1,OBSTA) !(T-1,0,HEADD T-1,1,OBSTA) !(T0,-1,HEADD T-1,-1,OBSTA) !(T1,0,HEADD T1,-1,OBSTA) !(T0,1,HEADD E-1,0 E1,0 E1,1 E-1,1 !(!T1,-1,OBSTA !T-1,-1,OBSTA)) !(T0,-1,HEADD E-1,0 E1,0 E-1,-1 E1,-1 !(!T-1,1,OBSTA !T1,1,OBSTA)) !(T1,0,HEADD E0,1 E0,-1 E1,1 E1,-1 !(!T-1,1,OBSTA !T-1,-1,OBSTA)) !(T-1,0,HEADD E0,-1 E0,1 E-1,-1 E-1,1 !(!T1,-1,OBSTA !T1,1,OBSTA))

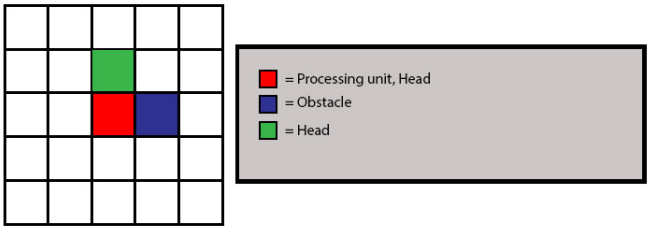

Figure 5.17: Regular head deactivation

- Check if there is a head in a more advanced position than the current module in a bridge created by the exit of a cycle (see Figure 5.18): `!(T0,1,HEADD T-1,1,STOPA E1,1 E1,0 E1,-1 !(!E-1,0 !T-1,0,ACTIV) !(!E-1,-1 !T-1,-1,ACTIV)) !(T0,-1,HEADD T1,-1,STOPA E-1,-1 E-1,0 E-1,1 !(!E1,0 !T1,0,ACTIV) !(!E1,1 !T1,1,ACTIV)) !(T1,0,HEADD E1,-1 T1,1,STOPA E0,-1 E-1,-1 T-1,0,STOPA !(!T0,1,ACTIV !E0,1) !(!T-1,1,ACTIV !E-1,1)) !(T-1,0,HEADD E-1,1 T-1,-1,STOPA E0,1 E1,1 T1,0,STOPA !(!T0,-1,ACTIV !E0,-1) !(!T1,-1,ACTIV !E1,-1))`

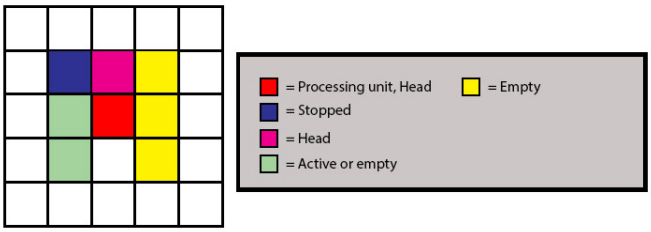

Figure 5.18: Head deactivation in certain cycle configurations

Action:
-Change state to stopped

5.3.10 Head readjustment

1 rule
There are situations where a cycle closes, that temporally the head is not on its proper location. This rule is designed to move the head around the base of static modules that goes over the obstacles of the system following the right hand rule in order to reestablish the head in its correct position.

Precondition:
-Check if stopped state: `SSTOPA`
- Check if stopped in an advanced position than the head in a straight line (see Figure 5.19): `!(T0,-1,HEADD !(!T0,1,STOPA !T0,1,OBSTA`

```
!E0,1) T1,0,OBSTA) !(T0,1,HEADD !(!T0,-1,STOPA !T0,-1,OBSTA
E0,-1) T-1,0,OBSTA) !(T-1,0,HEADD !(!T1,0,STOPA !T1,0,OBSTA
!E1,0) T0,-1,OBSTA) !(T1,0,HEADD !(!T-1,0,STOPA !T-1,0,OBSTA
!E-1,0) T0,1,OBSTA)
```

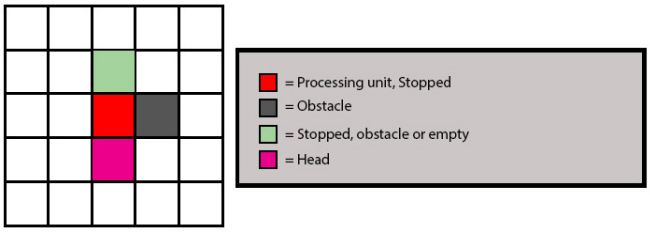

Figure 5.19: Head transmission over the base

- Check if stopped in an advanced position than the head in a corner (see Figure 5.20): `!(!(!T1,0,STOPA !T1,0,OBSTA !E1,0) T0,-1,HEADD T1,-1,OBSTA) !(!(!T-1,0,STOPA !T-1,0,OBSTA !E-1,0) T0,1,HEADD T-1,1,OBSTA) !(!(!T0,-1,STOPA !T0,-1,OBSTA !E0,-1) T-1,0,HEADD T-1,-1,OBSTA) !(!(!T0,1,STOPA !T0,1,OBSTA !E0,1) T1,0,HEADD T1,1,OBSTA)`

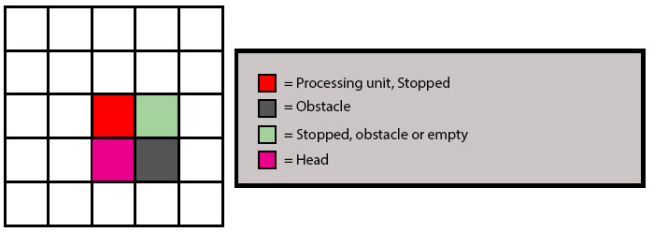

Figure 5.20: Head transmission over coners in the base

- Check if stopped is in an advanced position against a head in a bridge created in a cycle (see Figure 5.21): `!(T0,-1,HEADD E-1,0 E-1,-1 E1,0 E1,-1 E1,1 T0,1,STOPA T-1,1,STOPA T0,-2,STOPA) !(T0,1,HEADD E1,0 E1,1 E-1,0 E-1,1 E-1,-1 T0,-1,STOPA T1,-1,STOPA T0,2,STOPA) !(T-1,0,HEADD T1,0,STOPA E-1,1 E0,1 T1,1,STOPA E-1,-1 E0,-1 E1,-1 T1,1,STOPA T-2,0,STOPA) !(T-1,0,HEADD T-1,0,STOPA E1,-1 E0,-1 T-1,-1,STOPA E1,1 E0,1 E-1,1 T-1,-1,STOPA T2,0,STOPA)`

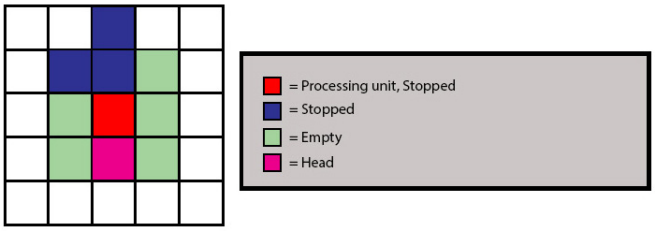

Figure 5.21: Head transmission in certain types of cycle configuration

- Check if stopped is in an advanced position against a head at the end of a bridge created in a cycle (see Figure 5.22): `!(T-1,0,STOPA T0,-1,HEADD T0,1,STOPA !(!E-1,1 !T-1,1,STOPA) T1,1,OBSTA E1,0 E1,-1 E1,-2 T0,-2,STOPA !(!T-1,-1,ACTIV !E-1,-1) !(!T-1,-2,ACTIV !E-1,-2)) !(T1,0,STOPA T0,1,HEADD T0,-1,STOPA !(!E1,-1 !T1,-1,STOPA) T-1,-1,OBSTA E-1,0 E-1,1 E-1,2 T0,2,STOPA !(!T1,1,ACTIV !E1,1) !(!T1,2,ACTIV !E1,2)) !(T-1,0,STOPA T-1,1,OBSTA !(!E-1,-1 !T-1,-1,STOPA) T0,-1,STOPA E0,1 E1,1 T1,0,HEADD E2,1 T2,0,STOPA !(!T1,-1,ACTIV !E1,-1) !(!T2,-1,ACTIV !E2,-1)) !(T1,0,STOPA T1,-1,OBSTA !(!E1,1 !T1,1,STOPA) T0,1,STOPA E0,-1 E-1,-1 T-1,0,HEADD E-2,-1 T-2,0,STOPA !(!T-1,1,ACTIV !E-1,1) !(!T-2,1,ACTIV !E-2,1))`

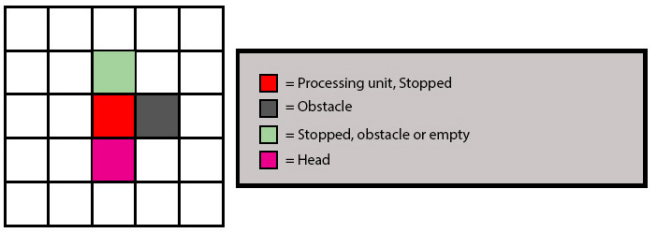

Figure 5.22: Head transmission in other type of cycle configuration

5.3.11 Superactive movement

8 rules
A superactive can move in situations where an active module cannot. Since the superactive activation only happens in punctual situations where no other modules can move, only checking if the movement is possible and that there are no active modules nearby is enough to decide to move. Once a superactive has moved it becomes active. Superactive movements are intended to break wall deadlock configurations not to be a strategy of locomotion.

There are two types of superactive movement: linear or diagonal

Straight movement rule of superactive modules, it corresponds to the 'Superactive N movement' rule (see Figure 5.23):

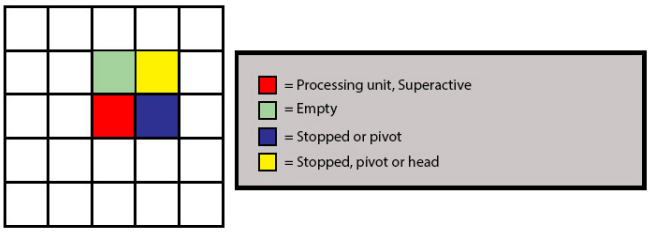

Figure 5.23: Check if superactive can move, N movement

Diagonal movement rule of superactive modules, it corresponds to the 'Superactive NE movement' rule:

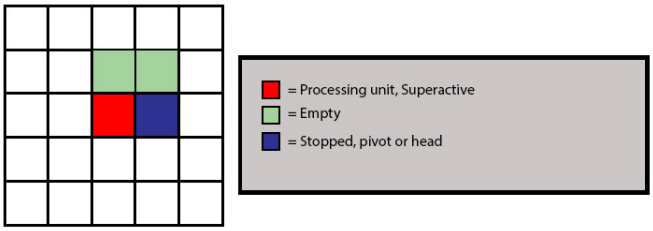

Figure 5.24: Check if superactive can move, NE movement

5.3.12 Superactive detection

1 rule
When a pivot is detected in a situation that characterizes a deadlock, it becomes superactive in order to move using less restrictive movement rules and break the deadlock allowing the system to move again.

Precondition:
- Check if pivot: `SPIVOT`
- Check if diagonal deadlock in any direction (see Figure 5.25): `!(E0,-1 T1,0,STOPA (!T0,1,OBSTA !T0,1,ACTIV !E0,1) !(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!(T1,-1,OBSTA) !(E1,-1 T2,0,STOPA T2,-1,OBSTA))) !(E0,1 T-1,0,STOPA (!T0,-1,OBSTA !T0,-1,ACTIV !E0,-1) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD) !(!(T-1,1,OBSTA) !(E-1,1 T-2,0,STOPA T-2,1,OBSTA))) !(E1,0 T0,1,STOPA (!T-1,0,OBSTA !T-1,0,ACTIV !E-1,0) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!(T1,1,OBSTA) !(E1,1 T0,2,STOPA T1,2,OBSTA))) !(E-1,0 T0,-1,STOPA (!T1,0,OBSTA`

```
!T1,0,ACTIV !E1,0) !(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD)
!(!(T-1,-1,OBSTA) !(E-1,-1 T0,-2,STOPA T-1,-2,OBSTA))
```

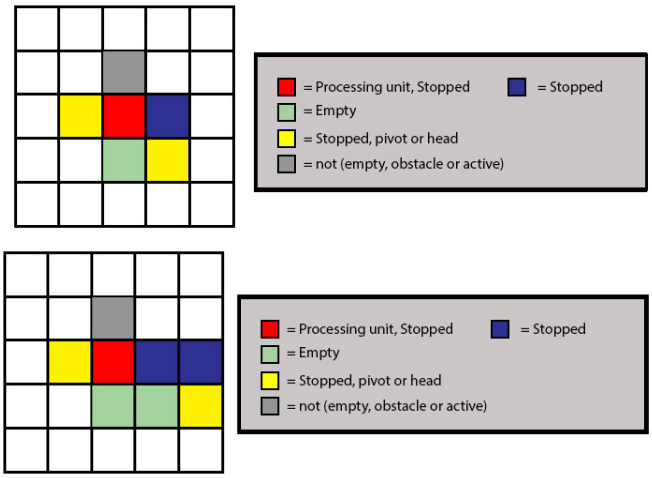

Figure 5.25: Check if there is a wall deadlock configuration

- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`

- Check if no actives on its 16 second level direct neighbors (in order to avoid to become superactive): `!T0,2,ACTIV !T1,2,ACTIV !T2,2,ACTIV !T2,1,ACTIV !T2,0,ACTIV !T2,-1,ACTIV !T2,-2,ACTIV !T1,-2,ACTIV !T0,-2,ACTIV !T-1,-2,ACTIV !T-2,-2,ACTIV !T-2,-1,ACTIV !T-2,0,ACTIV !T-2,1,ACTIV !T-2,2,ACTIV !T-1,2,ACTIV`

Action:

-Become superactive.

5.3.13 Perfect detection

1 rule

When a pivot or stopped module finds itself in the exit position of a perfect cycle, it becomes a perfect cycle beaker which is used to disassemble perfect deadlocks.

Precondition:

- Check if module is active or pivot: `!(!SSTOPA !SPIVOT)`

- Check if the module is in any of the special configurations characteristic of a perfect cycle end (see Figure 5.26): `!(E0,1 E0,-1 E1,1 !(!E-1,-1 !(!(!T-1,-1,PIVOT !T-1,-1,STOPA) !(!T-2,-1,OBSTA !T-2,0,OBSTA))) T1,0,HEADD !(!T-1,0,STOPA !T-1,0,PIVOT)) !(E0,-1 E0,1 E-1,-1 !(!E1,1`

```
!(!(!T1,1,PIVOT !T1,1,STOPA) !(!T2,1,OBSTA !T2,0,OBSTA)))
T-1,0,HEADD !(!T1,0,STOPA !T1,0,PIVOT)) !(E1,0 E1,-1 E-1,0
!(!E-1,1 !(!(!T-1,1,PIVOY !T-1,1,STOPA) !(!T-1,2,OBSTA
!T0,2,OBSTA))) T0,-1,HEADD !(!T0,1,STOPA !T0,1,PIVOT))
!(E-1,0 E-1,1 E1,0 !(!E1,-1 !(!(!T1,-1,PIVOY !T1,-1,STOPA)
!(!T1,-2,OBSTA !T0,-2,OBSTA))) T0,1,HEADD !(!T0,-1,STOPA
!T0,-1,PIVOT)) !(E0,1 E1,1 E1,-1 !(!T-1,0,OBSTA !T-1,1,OBSTA)
T1,0,HEADD !(!T0,-1,STOPA !T0,-1,PIVOT)) !(E0,-1 E-1,-1
E-1,1 !(!T1,0,OBSTA !T1,-1,OBSTA) T-1,0,HEADD !(!T0,1,STOPA
!T0,1,PIVOT)) !(E1,0 E1,-1 E-1,-1 !(!T0,1,OBSTA !T1,1,OBSTA)
T0,-1,HEADD !(!T0,1,STOPA !T0,1,PIVOT)) !(E-1,0 E-1,1 E1,1
!(!T0,-1,OBSTA !T-1,-1,OBSTA) T0,1,HEADD !(!T0,-1,STOPA
!T0,-1,PIVOT)))
```

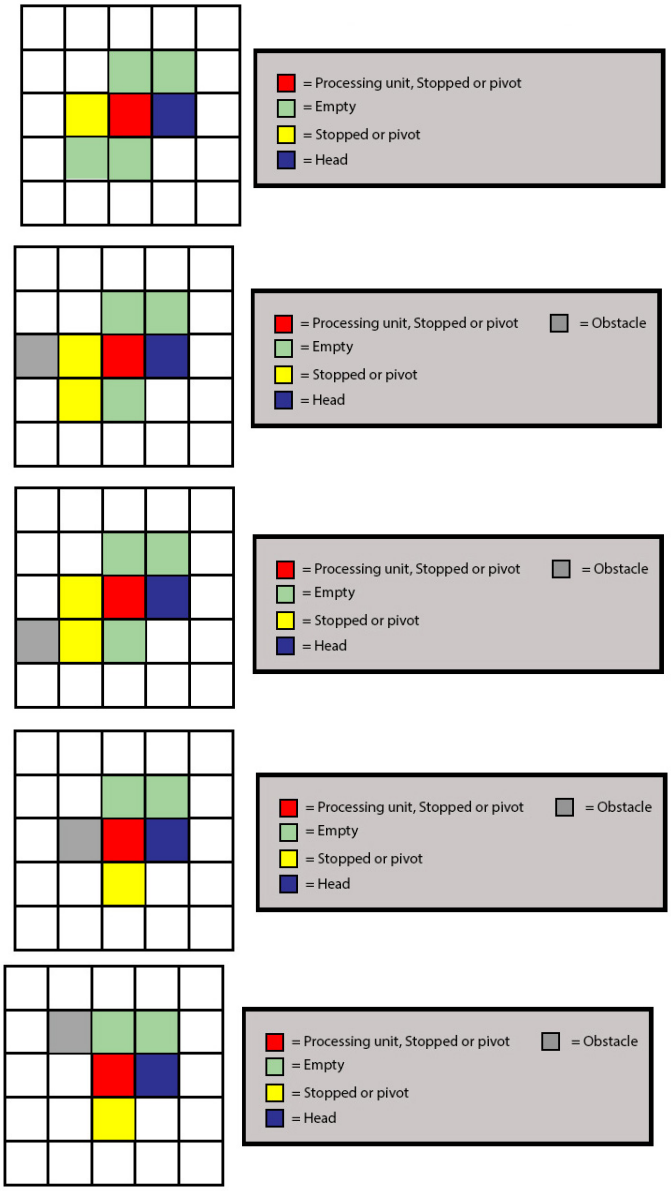

Figure 5.26: Perfect cycle breaker conditions

- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`

Action:

-Become perfect module

5.3.14 Perfect movement

1 conceptual rule implemented in 4 rules
A perfect module can only move diagonally over a head module following the right hand rule, it has the same rule as the standard diagonal movement but without the collision checks and the obligation to be a leaf. After the movement, the module becomes active (see Figure 5.27).

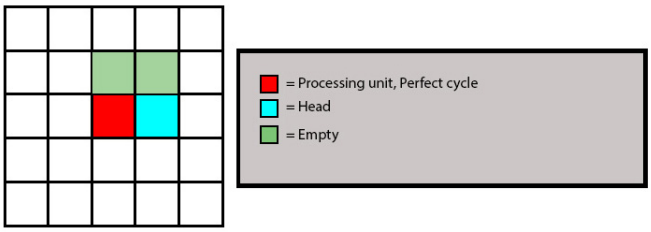

Figure 5.27: Check if movement can be done by a prefect cycle, NE movement

5.3.15 Perfect deactivation

1 rule
When a perfect module looses the head in its four direct neighbor, it becomes stopped.

5.3.16 T structure

2 conceptual rules implemented in 16 rules
Tailed cycles have always in its exit a characteristic T formed by the joint of two parts of the base of static modules of the system. Each T has only a correct direction of disassemble and a distinctive configuration. This rules focus on identifying the T module and also the direction of movement of it. To implement this concept, I use 8 different states to identify the directions of movement and use those states to induce the correct movement.

T detection in straight direction explained more in depth, it corresponds to the 'N T structure' rule:

Precondition:

- Check if the module is a pivot or a stopped module: `!(!SPIVOT !SSTOPA)`

- Check if exist the three characteristic configurations of a T having a module of disassemble in north direction (see Figure 5.28): `!(!(!(!T-1,0,STOPA !T-1,0,PIVOT) E0,1 E0,-1 !(!T1,1,STOPA !T1,1,HEADD) T1,0,STOPA !(!T1,-1,STOPA !T1,-1,PIVOT) T2,1,OBSTA !(!T2,0,STOPA !T2,0,PIVOT !T2,0,OBSTA`

```
!E2,0) !(!T2,-1,STOPA !T2,-1,PIVOT !T2,-1,OBSTA !E2,-1))
!(!(!T-1,0,STOPA !T-1,0,PIVOT) E0,1 E0,-1 E1,-1 T1,0,STOPA
!(!T1,1,STOPA !T1,1,HEADD) T2,1,OBSTA !(!T2,0,STOPA
!T2,0,PIVOT) !(!T2,-1,STOPA !T2,-1,PIVOT !E2,-1))
!(T-1,0,OBSTA E0,1 !(!T0,-1,STOPA !T0,-1,PIVOT) !(!T1,1,STOPA
!T1,1,HEADD) T2,1,OBSTA T1,0,STOPA E1,-1 T2,0,STOPA !(!E2,-1
!T2,-1,STOPA !T2,-1,PIVOT)))
```

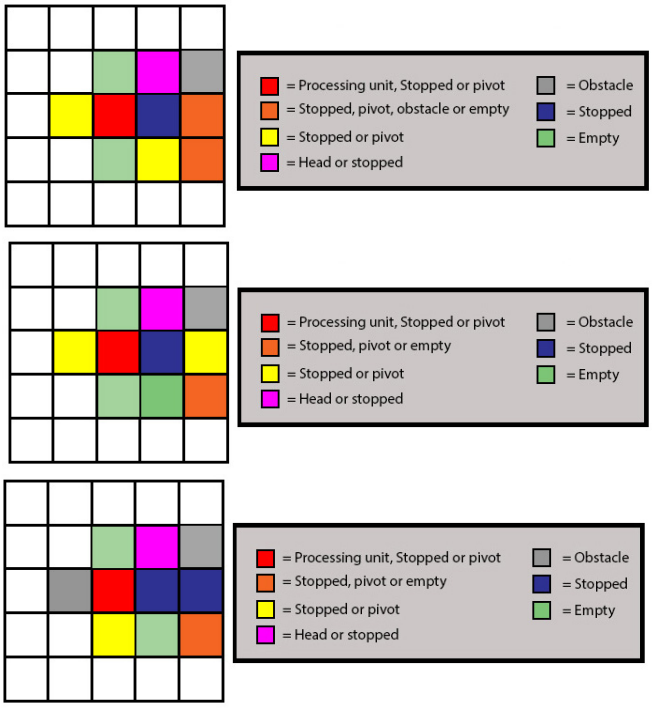

Figure 5.28: T structure conditions in straight direction, N T structure

- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`

Action:

- Become north tailed cycle breaker

T module movement in straight direction explained more in depth, it corresponds to the 'NCYCL movement' rule:

Precondition:

- Check if the module is a north tailed cycle breaker: `SNCYCL`
- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`

Action:
- Move to north direction and become active
* Note that we already know to which direction to move and just checking that there are no active modules is enough to decide to move.

T detection in diagonal direction explained more in depth, it corresponds to the 'NE T structure' rule:
Precondition:
- Check if the module is a pivot or a stopped module: `!(!SPIVOT !SSTOPA)`
- Check if exist the three characteristic configurations of a T having a module of disassemble in north-east direction (see Figure 5.29): `!(!T-1,0,STOPA !T-1,0,PIVOT) E0,1 E0,-1 E1,1 !(!T1,0,STOPA !T1,0,PIVOT) !(!T1,-1,STOPA !T1,-1,PIVOT) !(!T2,1,STOPA !T2,1,PIVOT !E2,1) T2,0,STOPA !(!T2,-1,OBSTA !T2,-1,STOPA)`

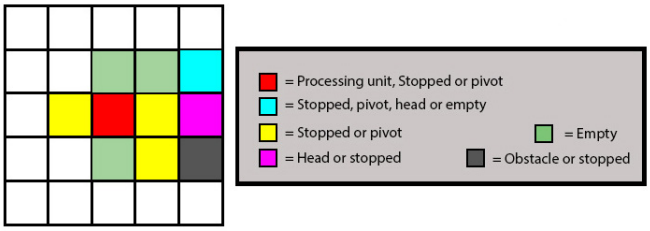

Figure 5.29: T structure conditions in diagonal direction, NE T structure

- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`
Action:
- Become north-east tailed cycle beaker

T module movement in diagonal direction explained more in depth, it corresponds to the 'NCYCL movement' rule:
Precondition:
- Check if the module is a north-east tailed cycle breaker: `SNECYC`
- Check if no actives on its 8 direct neighbors: `!T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV`
* Note that we already know to which direction to move and just checking that there are no active modules is enough to decide to move
Action:
- Move to north direction and become active

Chapter 6

Project management

6.1 Initial milestone

At the early stages of the project, we did a complete analysis and planning of the workload and tasks that would involve the project. Here are presented those initial analysis. In the next section of this chapter we will do the final milestone where we contrast and compare the final results and details against the initial milestone.

6.1.1 Obstacles and risks

Not finding any interesting result
It is possible that during the development of the project, no clever strategy or considerable improvement is found as the problem engaged is considerably complex. If such situation appears during the project, some alternatives could be taken as the field of study is large enough to modify the development of the project (3D approach, hexagonal modules, synchronous movement, ...).

Tight schedule
The project is intended to be completed in four months. As the amount of work is overwhelming and at the same time the project developer has to deal with two degree courses, there exists a high risk of missing the deadlines. In order to prevent this situation, adviser and student have scheduled weekly meetings to track the progress of the project.

Simulation issues
It can happen that once we have a promising distributed algorithm, the simulator does not recognize a certain strategy or the modules are not initially thought to be programmed in that manner. If we face this situation we could modify the simulator since we have access to its source code.

6.1.2 Methodology and rigor

To ensure the success of this project, a strict and efficient organization of the tasks is the key to obtain good results. To manage all the work to do, we will divide the workload in tasks and small and focused objectives that together will build the whole project. An agile methodology suits perfectly the project since it is intended to be finished in a short period of time where fast results are primordial. The followed methodology will be inspired in the SCRUM methodology.

In a nutshell, the project will be split into several phases: the reading phase, the analysis phase, the thinking phase and finally the implementation and validation phase (see Section 6.1.3). All this phases will correspond to the iterations of the agile methodology. Each iteration will aim to finish its dedicated objective under the supervision of the project adviser who will decide when the results are good enough to proceed to the next phase.

Monitoring tools

To keep track of the organization and evolution of the project We will use the software Trello [4]. Trello allows to teams to guarantee the correct follow-up of a project giving a platform to organize the workload in tasks and giving tools to state and control the progress of them. I will create a team with me and my project adviser and also plan weekly meetings to evaluate the results and track the project development face to face.

6.1.3 Temporal planning

This project was started on February the 1st and is intended to be finished the 29th of June.

The project involves a huge workload that ideally has to be completed in roughly four months. In order to achieve this goal, a carefully organized action plan needs to be scheduled. That schedule has to include all the tasks and also understand all the phases by which the project will go through to succeed. Additionally, some alternatives and troubleshooting strategies will be contemplated in the action plan in order to control possible setbacks.

Phases and tasks

Bureaucracy and planning This is one of the first stages that any sort of project must do. In my case, I had to submit a project proposal to the chief of the Computer Science department describing the nature of my project. The inscription was done after I had read all the documentation about the final degree project and found a motivating project to work on.

Afterwards, during the early stages of the project, I had to decide with my project adviser an action plan and a feasible planning. Those decisions

were taken after the first stages of reading some documentation about the subject in order to have in mind the work that I will be performing the following months.

Reading Having an accurate mental picture of the engaging problem magnitude in which the project focuses is imperative to success. This project phase is devoted to that. It involves the reading of the most recent and relevant articles and publications related to the field of study, covering all the dimensions of the project, in order to obtain a deep knowledge of the subject and build a solid base to start working on.

Analysis Once all the significant documentation is reviewed, it has to be decided what will exactly be done. That means we need to decide from which angle attack the problem and, more important, learn from useful techniques that have succeeded before while avoiding mistakes and wrong approaches that were previously done. The idea is to wisely benefit from the previous work done on the problem and lead the solving process towards a genuine resolution.

This part of the project requires a good intuition and fruitful debates between the student and the adviser to glimpse the shape that a possible solution might have. It is crucial to make a great decision to orientate the project towards successful results.

Crafting and designing ideas All the creative and inventive of the project is done in this phase. Basically here is where the improvements on current solutions or the creation of new algorithms occur. Making use of all the knowledge gathered from the previous phases and using imagination and intuition alongside theoretical testing is fundamental. Therefore, here will be explored new paths to try to reach some relevant conclusions.

Implementation and simulation After having achieved promising theoretical locomotion strategies, the phase of implementation shall start to test if the ideas are useful on real situations. The implementation will be done in a simulator provided by the project adviser [7]. That digital environment allows us to simulate the execution of distributed algorithms on an asynchronous virtual reality. Several tests will be performed facing an arbitrary lattice-based modular robot against complex obstacles to see the performance of the implemented strategies.

If in this phase the implementation produces disappointing results, the project will step back to the previous designing phase to rethink the algorithm.

Formal proof This phase consists in proving mathematically the correctness of the distributed algorithm or improvement that has been proposed. The proof will aim to confirm that the proposed strategies will always work under certain circumstances. Since implementing a set of rules and discarding early wrong results is more agile doing simulations against complex obstacles, the formal proof will initially be postponed to be done after some tests in the simulator. However, depending on the nature of the proposed algorithms it could be perfectly done before the implementation.

Documenting the project The task of documenting will be split into two major blocks. The objective is to create a final complete report.

Firstly, all the documentation related with the state of the art and the description of the project, alongside economical and temporal budgets, will be done on the early stages of the project. Such work will be collected from the deliverable documents that have to be presented to the project management module (GEP) of the graduate thesis.

Finally, after having achieved satisfactory results on the project, all the conclusions will be gathered and added to the final report. The document will also be filled with all the observations done in working stages and procedures that have been followed.

Presentation The last part of the project will involve a presentation in front of a committee of experts that will evaluate the quality of the work. This phase will need the creation of some slides to backup the presentation and rehearsal to prepare a good performance.

6.1.4 Schedule

Table 6.1: Expected duration of tasks

Task or phase	**Time spent (hours)**
Bureaucracy and planning	10
Reading	75
Analysis	25
Crafting and designing ideas	175
Implementation and simulation	75
Formal proof	30
Documenting the project	150
Presentation	10
Total	550

Time table

6.1.5 Gantt chart

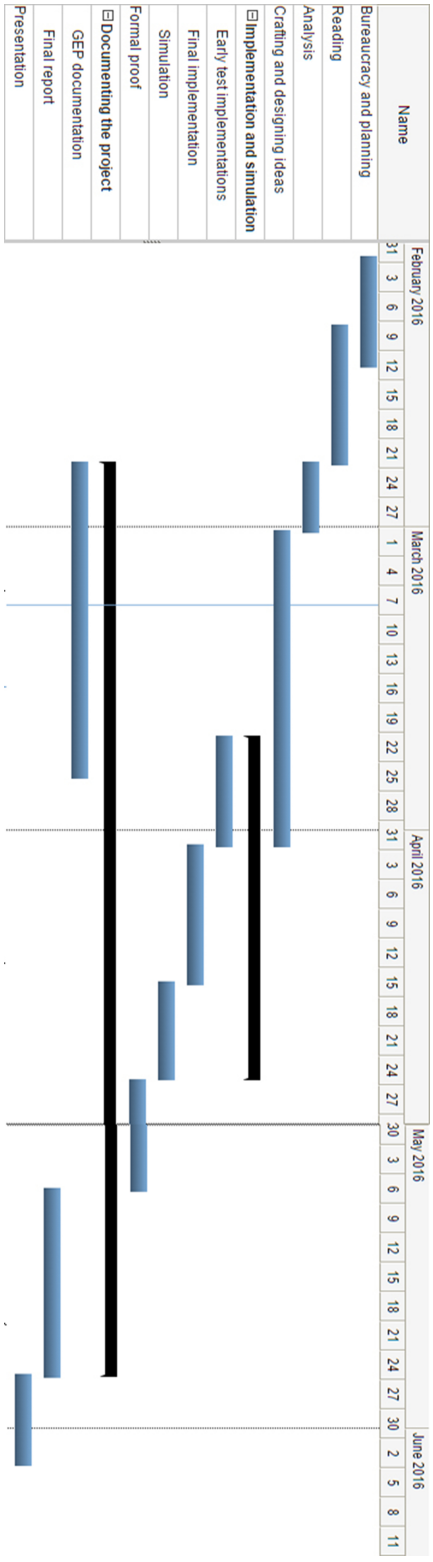

Figure 6.1: Gantt chart showing the planned schedule

6.1.6 Resources

Hardware This project works in a theoretical dimension so not much dedicated hardware is needed to finish it. Just a regular computer to perform all the reading, simulation and implementation is enough. The use of hardware resources will be primary used in the phase of reading, implementation and documenting. In other phases the hardware resources will be just used as eventual support.

Software To accomplish certain tasks, some specific software will be used.

- **Simulator**[7] Java application that allows the creation of environments where a lattice based modular robot can be programmed and then simulate its behavior. This software is provided by the project adviser. This tool will be exclusively used in the implementation phase.

- **LaTeX** Typesetting system used for creating and formatting documents widely used on academic publications. This tool will be used to write and format the report. Documenting and creating the presentation will be the tasks where this resource is used.

Human To accomplish this project a huge amount of human resources will be used. There is not a need of a vast diversity of individuals to work on the project, however, the quantity of hours invested on the project from the people engaged will be enormous. Just with two people formed in in the field of Computer Science and Mathematics will be enough. One person will perform the role of project adviser while the other will be the project developer. The project adviser will influence all the phases, nevertheless, its influence in the phases of analysis and formal proof will be crucial. The project developer will carry out most of the workload of the project.

Action plan

The action plan is composed by the tasks mentioned before in this subsection and will follow the schedule presented in the gantt chart (see Figure 6.1). There will also be weekly meetings with the project adviser to track the project progress and discuss eventual issues that may appear during the different stages.

Alternatives Ideally the project will follow the scheduled plan. However, if any major obstacle is encountered and the project gets stuck, the goal of the project would be adjusted (see Section 6.1.1). As all alternative goals are just variations on the proposed problem, there would be no need to make big changes on the schedule.

6.1.7 Budget estimation

The budget estimation will be divided in four sections in order to present the information organized in a simple and comprehensive way. The last section will have the sum of all costs plus the application of taxes.

The useful life and the project's estimated duration will be used to calculate the amortization. As mentioned in previous sections, the project is initially intended to be completed in four months.

All the elements that will appear in the budget come from all the tasks and phases mentioned in the Gantt chart (See Figure 6.1).

Hardware resources The following costs will be produced by hardware resources during the project development.

Product	Price (€)	Useful life (years)	Amortization (€)
Computer	1500.00	4	187.50
Total	1500.00 €		187.50 €

Table 6.2: Hardware costs

Hardware resources will be used in all the phases of the project, nevertheless in the reading, documenting and specially in the implementing phase will be the principal working tool.

Software resources These will be the costs generated by the usage of software resources. As all software products used are free, no cost at all is generated by this group of resources.

Product	Price (€)	Useful life (years)	Amortization (€)
Windows 10	0.00	5	0.00
LaTeX	0.00	5	0.00
Simulator	0.00	2	0.00
Total	0.00 €		0.00 €

Table 6.3: Software costs

Like hardware resources, software resources will be used in all the phases of the project. Remark that the Simulator will be used in the implementing phase and Latex in the documenting tasks.

Human resources All humans involved in the project will generate the following expenses. In this case the role of computer scientist and software implementator will be assumed by me.

Product	Hours	Price per hour	Cost (€)
Project adviser	100.00	50.00 €/hour	5000.00
Computer scientist	350.00	40.00 €/hour	14000.00
Software engineer	200.00	35.00 €/hour	7000.00
Total	650.00		26000.00 €

Table 6.4: Human costs

Human resources will be equally distributed among all tasks. The analysis and formal proof phases will require huge collaboration between the project developer and the project adviser. The creating phase will require an intense dedication and so a bigger amount of human resources among other resources. The price per hour of each different role present in the project have been estimated regarding data taken from the webpage payscale.com.

Indirect costs There will also be costs indirectly generated by the activities involved in the project.

Product	Price	Quantity	Cost (€)
Electricity	0.07€/Kwh	7142.85 Kwh	500.00
Notebook	5.00 €/unit	1 unit	5.00
Paper	25.00 €/pack	1 pack	25.00
Pen	1.50 €/unit	2 units	3.00
Total			533.00 €

Table 6.5: Indirect costs

Indirect costs are associated with the consumption of other resources and for so their distribution is balanced in the project schedule.

Total budget Here are all the estimated costs of the project. After the clean sum of the costs, there are applied the taxes giving the real total budget of the project.

Budget	**Cost**
Hardware resources	187.50 €
Software resources	0.00 €
Human resources	26000.00 €
Indirect costs	533.00 €
Total	26720.50 €
Taxes	21%
Total with taxes	32331.81 €

Table 6.6: Total budget

Budget control

The previous section shows an accurate picture of the distribution that costs have in this project. It is undeniable that the investment in human resources is much bigger than the overall sum of hardware, software and indirect costs. That fact is due to the nature of the project where most of the tasks are theoretical and done in the dimension of the ideas. Regarding to that, the proposed strategies to control the budget will mainly focus on human activities.

It is easy to see that the total cost of human resources depends directly on the duration of the project. In order to reduce expenses, the project organization and work should be as efficient as possible. To achieve that goal and efficient temporal planning plays a capital role: the distribution of tasks in the Gantt chart (see Section 6.1.5) tries to do that. Moreover, the weekly meetings between the project developer and the adviser are thought to be the mechanism of control and tracking of the progress. With those tools the amount of time invested on the project will be controlled, and so the project budget.

6.2 Final milestone

Having finished the tasks of the project we will analyse all the points presented in the initial milestone and review its performance during the development of the project.

6.2.1 Obstacles and risks

Following the work plan that was initially designed all the obstacles and risks where avoided accomplishing the goals that were set in the scope. Moreover, all the workload was done in the time frame that was predicted and the implementation was successfully coded in the simulator.

6.2.2 Methodology and rigor

The chosen agile methodology based in the SCRUM method suited perfectly the tasks and provided a dynamic framework that induced a correct development of the project.

Monitoring tools

The software Trello turned out to be a useful tool to keep track of the development of the project. However, at the end of story, the weekly meetings between the student and the project adviser keep up the progress of the project tracked leaving behind Trello in a second plane.

6.2.3 Temporal planning

The project followed the initial temporal plan. No initial task was changed and just minor readjustments had to be done in order to finish the project. Those little changes were done in the implementation phase which took longer than initially expected. In the following subsections we will explain how each task was developed and show a final count of the hours invested in each task

Phases and tasks

Bureaucracy and planning This task was very simple to accomplish and did not take too much time of the project.

Reading A lot of documentation was provided by the project adviser composed by some relevant previous publications on the field of modular robots and other similar projects. All that information was read and contributed with important ideas that influenced further stages. This phase took a similar time to accomplish as predicted.

Analysis From all the notes and ideas gathered in the previous tasks, we discarded the ones that seemed not very promising and keep with the ones that were more promising. Here there was decided if the project would try to improve current strategies or seek for a possible general algorithm, which was finally the decision made. This task took as much time as in the early estimation.

Crafting and designing ideas In this task different approaches to the solution and different strategies of locomotion arose. The initial ideas ended up to not work very well in the general case. Nevertheless, a powerful idea come up surprisingly early and inspired the final solution. In a third of the initial expected time this task was closed.

Implementation and simulation Here most of the project workload was done. The implementation of the algorithm idea was not very complicated. In a small amount of time there was an early implementation that worked in basic cases. However, the problem show up to be more complicated than expected. In the testing phase we come up with some complex degenerated obstacles where the first implementation failed. From this situation the initial algorithm was submit under remodeling until a final result that solved the general problem of locomotion was formulated. That final algorithm was based on the first idea but had also incorporated multiple considerations for degenerated obstacles leaving a strategy with an implementation with more than a hundred rules. Since that solution was extremely complex and

seemed very hard to proof its correctness, a couple of weeks were invested in simplifying the algorithm and studying the obstacles in order to find the core of its nature. Finally a simpler algorithm was crafted with 18 conceptual rules and implemented in 50 simple rules. All this process took much more than initially expected, but, since the progress was ahead of the initial plan at the start of this task, the project did not fall back on deadlines.

Formal proof Since the algorithm obtained ended up having a consistent idea behind it with very powerful and simple rules, the proof of correctness and complexity analysis were fairly simple to formulate. The time invested in this task was the initially intended.

Documenting the project The documentation of the project was split in two parts. Having all the initial analysis of the project and its management written down before working on it. The second part was done at the end of the project, when the goal was reached. The whole task took more or less the initially expected time.

Presentation This last part of the project consisted in the creating of some slides and a few sessions of rehearsal in order to be finished. The time it took to be accomplished was the initially expected.

6.2.4 Time table

Here are presented the real duration of all the tasks of the project. It can be seen that, unless few tasks, the initial prediction was accurate.

Table 6.7: Duration of tasks

Task or phase	**Time spent (hours)**
Bureaucracy and planning	5
Reading	80
Analysis	30
Crafting and designing ideas	50
Implementation and simulation	200
Formal proof	10
Documenting the project	150
Presentation	10
Real total	535
Estimated total	550

6.2.5 Used resources

Regarding resources, the initially proposed use of resources was what ended up to be needed in the development of the project. No extra software,

hardware or human resources where needed since no unexpected task or help was needed in order to obtain the results.

6.2.6 Final budget

The final budget did not changed significantly from the initial estimated budget since the project was finished in the expected time and no extra tasks or resources where needed in order to finish it. Leaving an overall cost of 32331.81 €.

6.3 Sustainability

6.3.1 Economic sustainability

All human and material costs are detailed this document alongside the unexpected events planning (see Section 6.1.3). Updates and maintenance of the generated results are null since the product is an algorithm. The costs of the project are viable and competitive. This can be stated because of the efficient plan that has been designed to take the most advantage of the working hours and resources. A reduction on working hours would have resulted counterproductive as the quality of the results would have been affected: the creative process needs its time to be performed and a reduction of time in testing or proving would add to the project the possibility of getting wrong results.

The project forms part of a bigger initiative created by international universities that aim to study and improve the field of distributed algorithms for modular robots.

This project is going to get a mark of 7 out of 10 in the economical viability area since its price is affordable and well adjusted but the direct economical benefits from it are not very impressive. That is because of the initial phase where lattice-based modular robots are: nowadays the commercial or economical benefits of any advance of them are despicable.

6.3.2 Social sustainability

The nature of this project is purely academic. That means that the results obtained from the project will not appear in the market or societies in short term. They will be included in the current front line results of the field and nourish further research. They might also be used in some testing prototypes or real simulations. The theoretical dimension of the results is due to the fact that lattice-based modular robots are relatively young in the field of Robotics, being machines that have to be vastly improved.

If long therm consequences are considered, the social implications of the project start to be more relevant. Lattice-based modular robots represent a

technology that promises to revolutionize the future. Their applications will vastly benefit societies; from medicine using nanotechnology to even spatial exploration. Therefore, a useful and efficient locomotion strategy is primal to have good performance. Those final locomotion strategies could greatly benefit from the results of this project, giving them a very high importance in the social ambit.

One of the main features of lattice-based modular robots is their versatility, giving them the possibility of working in almost every environment. That feature makes them perfect for exploring dangerous environments in emergencies or replacing humans in harming or dangerous tasks.

Regarding the previously mentioned points, this project is going to be awarded with a 7 in social sustainability since the results of this work could immensely help future societies but they will not directly help our actual society.

6.3.3 Environmental sustainability

The environmental impact of the project in nature is very small. The only aspect that represents a threat to the environment comes from the indirect consequences of the use of a computer. That is because of the electricity generation processes and the pollution generated in the computer manufacturing. However, those impacts are tiny and do not exceed the average environmental impact of a regular computer user. Moreover, a great part of the project consists on thinking and designing algorithms, activities that do not interfere in the integrity of nature.

The versatility of lattice-based modular robots could in a future replace lots of dedicated machines that nowadays perform specific tasks giving them a huge useful and reusable life saving the environment from crafting tons of machines that have a high obsolescence.

Having in mind that the project development will merely affect the environment integrity and that the results will help indirectly the environment in the future, the project will get a 9 in environmental sustainability.

6.4 Actors and stakeholders

Project developer
Only one person will be working in this project, that is myself. All the reading, thinking, implementation and documentation will be done by me, under the supervision of my adviser.

Project adviser
This project is directed by Vera Sacristán from the Mathematics department at UPC. Her role will be the guidance of the project and supervision of all

the results.

Mathematicians and computer scientists
This is a passive set of people that will provide inspiration with the current ideas of the subject. Additionally, they will benefit from the work done in this project.

Robotic engineers
As the group mentioned before, this group of people will benefit directly from the results of this project, in this case, in the programming of their devices.

6.5 Technical competences

At the beginning of the project, several technical competences of the Bachelor degree in Computer Informatics where related to this project regarding the nature of the problem faced. Now, with the final results on the table, we can evaluate how well the performance in those competences was.

- CCO1.1: 'To evaluate the computational complexity of a problem, to know algorithmic strategies that can provide it a solution, and to recommend, develop and implement the one that guarantees the best performance regarding the established requirements.'

  This competence was extremely used during the development of the project. It can be seen in the results that a functional algorithmic solution to the engaged problem was crafted. Moreover, a complexity analysis and a proof of correctness were formulated to support that solution showing a deep knowledge of development of this competence.

- CCO2.2: 'Ability for acquiring, obtain, formalize and represent human knowledge in a computational way for then solve problems using computer systems, particularly the ones related with computational aspects, perception, and actuation in intelligent environments.'

  Since the proposed algorithm of locomotion works for any given set of obstacles, it has to understand completely the environment in which the modular robot is deployed and make decisions intelligently at each step. Human concepts as direction of movement or local awareness of morphology had to be formalized and translated to the geometrical framework in order to design the conditions to apply certain strategies of locomotion. For those reasons, our solution exploited this technical competence.

- CCO2.4: 'Demonstrate knowledge and techniques of computational learning: design and implement applications and systems that use

them, including the ones that are dedicated to the automatic extraction of information and knowledge from big volumes of data.'

One possible line of work that was initially proposed, was to design an algorithm that could extract data from its environment, learn about it, and build an optimal strategy of locomotion. Since the final solution relies in geometrical properties and shifted the project from the frame of computational learning, techniques from this competence were not needed in the resolution of the problem.

- CCO3.1: 'Implement critical code following the criteria of execution time, efficiency and security.'

  Alongside the algorithm, a complete functional implementation of it was also coded. This implementation was coded maximizing the efficiency and minimizing the execution time. Even more, the provided implementation was simplified as much as possible in order to help understand the algorithm as a complementary explanation. For those reasons this technical competence was really present on the project.

6.6 Additional material

This project document will be complemented with a file with the implementation called 'rules.txt' plus a folder called 'testing' full of different obstacle configurations organized in different levels of complexity.

Chapter 7

Conclusions

7.1 Presented results

The aim of this project was to create a general distributed algorithm algorithm of locomotion for square lattice-based modular robots. This goal has been accomplished with excellent results: the final result is a distributed algorithm that solves the problem in an elegant approach, generating a set of only 18 conceptual rules that work with seven different states, alongside an efficient running time linear in the distance to walk and the number of modules of the robot. This algorithm is presented with its proof of correctness, its complexity analysis and a detailed description with a real implementation. Another point worth mentioning is that the set of rules can work either in synchronous or asynchronous systems.

7.2 Further work

There are interesting extensions that can be done using as base the results of this thesis.

Rotational or squeeze movements The presented algorithm works for 2D square lattice-based modular robots which move using sliding movements, its extension and integration with rotational and squeeze movements is possible and could have interesting results. At first sight, it seems that the extension to squeeze movements would lead to a simpler algorithm and the extension to rotational would add bigger constraints of free space in order to perform a movement.

2D hexagonal locomotion The presented algorithm works for 2D square lattice-based modular robots, its extension to hexagonal lattice-based modular robots seems very direct.

3D cubic locomotion The presented algorithm works for 2D square lattice-based environment, its extension to the 3D dimension of cubic lattice-

based modular robots could be very useful for real robots. Such extension could greatly benefit from the strategies used in this algorithm, nevertheless the basic shape which would use the caterpillar principle of locomotion in 3D should be precisely chosen and analysed since the problems of disconnections and deadlocks seem much harder in 3D.

7.3 Personal reflection

This project has brought me a rich bag of valuable experiences in either academic and personal dimensions.

From the academic point of view, I have put to test most of the abilities that I have been acquiring in college during the last years. Also, I had the opportunity to work hand to hand with a very inspiring professor, my adviser, from who I have learnt the rigor and professionalism at the time of working.

Personally speaking, I felt how is to engage a real problem solving situation and test my creativity and perseverance. I learnt from the experience of carrying out a real project of research alongside all the implied tasks and proved myself that I am able to do it. This experience inspired and motivated me to continue my studies on Computer Science and gave me a realistic perspective that will influence my near future and professional career.

References

[1] Classical single-purpose robot. http://i.blogs.es/6cbd12/m-blocks-mit-2/1366_2000.jpg. [Online; accessed 24-March-2016].

[2] Lattice-based modular robot. http://www.mekkam.com/project-type/robots-industriales/page/2/. [Online; accessed 24-March-2016].

[3] Modular robot. http://www.ros.org/news/2010/04/robots-using-ros-modlabs-ckbots.html. [Online; accessed 24-March-2016].

[4] Trello. https://trello.com/. [Online; accessed 25-March-2016].

[5] Byoung Kwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *2008 IEEE International Conference on Robotics and Automation, ICRA*, pages 3149–3155, 2008.

[6] Zack J. Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *I. J. Robotic Res.*, 23(9):919–937, 2004.

[7] Suneeta Ramaswami Ferran Hurtado, Enrique Molina and Vera Sacristán. Generic distributed universal constant-force reconfiguration of 2D lattice-based modular robotic systems simulator. http://www-ma2.upc.edu/vera/AgentSystems/, 2015. [Online; accessed 28-February-2016].

[8] Robert Fitch and Zack J. Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *I. J. Robotic Res.*, 27(3-4):331–343, 2008.

[9] A. Hemmerling. *Labyrinth problems: labyrinth-searching abilities of automata*. Teubner-Texte zur Mathematik. Teubner, 1989. page 18.

[10] Keith Kotay and Daniela Rus. Efficient locomotion for a self-reconfiguring robot. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, pages 2963–2969, 2005.

[11] Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, and Satoshi Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *I. J. Robotic Res.*, 27(3-4):373–386, 2008.

[12] Lorena Lusso. *Locomotion of Self-Organizing Robots.* Master's thesis, Master of Science in Advanced Mathematics and Mathematical Engineering (MAMME), UPC, 2013.

[13] N.A. Lynch. *Distributed Algorithms.* The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 1996.

[14] S. Murata and H. Kurokawa. *Self-Organizing Robots.* Springer Tracts in Advanced Robotics. Springer Japan, 2012.

[15] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Auton. Robots*, 10(1):107–124, 2001.

[16] G.C. Sirakoulis and A. Adamatzky. *Robots and Lattice Automata.* Emergence, Complexity and Computation. Springer International Publishing, 2014.

[17] S. Wong, S. Zhu, and J. Walter. Unpacking a cluster of modular robots. *Proceedings of the 2015 International Conference on Parallel and Distributed Techniques and Applications*, 2015.

Appendix: Complete implementation

```
     %###### STATES ######%
_init_start_
SOBSTA GRAY
SSTOPA BLUE
SACTIV RED
SPIVOT PINK
SHEADD CYAN
SSUPER YELLOW
SPERFE MAGENTA
SNCYCL ORANGE
SSCYCL ORANGE
SECYCL ORANGE
SWCYCL ORANGE
SSECYC ORANGE
SNWCYC ORANGE
SSWCYC ORANGE
SNECYC ORANGE
_init_end_

     %##### RULES #####%

%-------------------------------
%%%%%%%%% Movements by active
%-------------------------------

N movement
3
SACTIV E0,1 !(!E-1,0 !T-1,0,OBSTA !T-1,0,ACTIV) !(!T1
    ,1,STOPA !T1,1,PIVOT !T1,1,HEADD !T1,1,SUPER !T1,1,
    PERFE !T1,1,SECYC !T1,1,NWCYC !T1,1,SWCYC !T1,1,
```

```
    NECYC !T1,1,WCYCL !T1,1,ECYCL !T1,1,SCYCL !T1,1,
    NCYCL) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,
    SUPER !T1,0,PERFE !T1,0,SECYC !T1,0,NWCYC !T1,0,
    SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,ECYCL !T1,0,
    SCYCL !T1,0,NCYCL) !(T1,0,STOPA T1,1,STOPA T-1,2,
    HEADD T0,2,ACTIV) !(T-1,0,OBSTA E0,2 T-1,2,HEADD T
    -0,3,ACTIV !(!T-1,3,STOPA !T-1,3,PIVOT))
P0,1 A1111

S movement
3
SACTIV E0,-1 !(!E1,0 !T1,0,OBSTA !T1,0,ACTIV) !(!T
    -1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,HEADD !T-1,-1,
    SUPER !T-1,-1,PERFE !T-1,-1,SECYC !T-1,-1,NWCYC !T
    -1,-1,SWCYC !T-1,-1,NECYC !T-1,-1,WCYCL !T-1,-1,
    ECYCL !T-1,-1,SCYCL !T-1,-1,NCYCL) !(!T-1,0,STOPA !
    T-1,0,PIVOT !T-1,0,HEADD !T-1,0,SUPER !T-1,0,PERFE
    !T-1,0,SECYC !T-1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC
     !T-1,0,WCYCL !T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,
    NCYCL) !(T-1,0,STOPA T-1,-1,STOPA T1,-2,HEADD T0
    ,-2,ACTIV) !(T1,0,OBSTA E0,-2 T1,-2,HEADD T0,-3,
    ACTIV !(!T1,-3,STOPA !T1,-3,PIVOT))
P0,-1 A1111

E movement
3
SACTIV E1,0 !(!E0,1 !T0,1,OBSTA !T0,1,ACTIV) !(!T1,-1,
    STOPA !T1,-1,PIVOT !T1,-1,HEADD !T1,-1,SUPER !T1
    ,-1,PERFE !T1,-1,SECYC !T1,-1,NWCYC !T1,-1,SWCYC !
    T1,-1,NECYC !T1,-1,WCYCL !T1,-1,ECYCL !T1,-1,SCYCL
    !T1,-1,NCYCL) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,
    HEADD !T0,-1,SUPER !T0,-1,PERFE !T0,-1,SECYC !T0
    ,-1,NWCYC !T0,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !
    T0,-1,ECYCL !T0,-1,SCYCL !T0,-1,NCYCL) !(T0,-1,
    STOPA T1,-1,STOPA T2,1,HEADD T2,0,ACTIV) !(T0,1,
    OBSTA T2,1,HEADD E2,0 T3,0,ACTIV !(!T3,1,PIVOT !T3
    ,1,STOPA))
P1,0 A1111

W movement
3
SACTIV E-1,0 !(!E0,-1 !T0,-1,OBSTA !T0,-1,ACTIV) !(!T
    -1,1,STOPA !T-1,1,PIVOT !T-1,1,HEADD !T-1,1,SUPER !
    T-1,1,PERFE !T-1,1,SECYC !T-1,1,NWCYC !T-1,1,SWCYC
```

```
    !T-1,1,NECYC !T-1,1,WCYCL !T-1,1,ECYCL !T-1,1,SCYCL
     !T-1,1,NCYCL) !(!T0,1,STOPA !T0,1,PIVOT !T0,1,
    HEADD !T0,1,SUPER !T0,1,PERFE !T0,1,SECYC !T0,1,
    NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,
    ECYCL !T0,1,SCYCL !T0,1,NCYCL) !(T0,1,STOPA T-1,1,
    STOPA T-2,-1,HEADD T-2,0,ACTIV) !(T0,-1,OBSTA T
    -2,-1,HEADD E-2,0 T-3,0,ACTIV !(!T-3,-1,PIVOT !T
    -3,-1,STOPA))
P-1,0 A1111

NE movement
3
SACTIV E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA !T-1,0,ACTIV)
    !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,SUPER !
    T1,0,PERFE !T1,0,SECYC !T1,0,NWCYC !T1,0,SWCYC !T1
    ,0,NECYC !T1,0,WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,
    NCYCL) !(T1,0,STOPA T0,2,HEADD T1,2,ACTIV E1,1)
     !(T0,2,HEADD E2,1 E1,2 T2,2,ACTIV !(!T1,3,STOPA !
    T1,3,PIVOT))
P1,1 A1111

SW movement
3
SACTIV E0,-1 E-1,-1 !(!E1,0 !T1,0,OBSTA !T1,0,ACTIV)
    !(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD !T-1,0,
    SUPER !T-1,0,PERFE !T-1,0,SECYC !T-1,0,NWCYC !T
    -1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL !T-1,0,ECYCL !
    T-1,0,SCYCL !T-1,0,NCYCL) !(T-1,0,STOPA T0,-2,HEADD
     T-1,-2,ACTIV E-1,-1) !(T0,-2,HEADD E-2,-1 E-1,-2 T
    -2,-2,ACTIV !(!T-1,-3,STOPA !T-1,-3,PIVOT))
P-1,-1 A1111

NW movement
3
SACTIV E-1,0 E-1,1 !(!E0,-1 !T0,-1,OBSTA !T0,-1,ACTIV)
     !(!T0,1,STOPA !T0,1,PIVOT !T0,1,HEADD !T0,1,SUPER
    !T0,1,PERFE !T0,1,SECYC !T0,1,NWCYC !T0,1,SWCYC !T0
    ,1,NECYC !T0,1,WCYCL !T0,1,ECYCL !T0,1,SCYCL !T0,1,
    NCYCL) !(T0,1,STOPA T-2,0,HEADD T-2,1,ACTIV E-1,1)
    !(T-2,0,HEADD E-1,2 E-2,1 T-2,2,ACTIV (!T-3,1,STOPA
     !T-3,1,PIVOT))
P-1,1 A1111

SE movement
```

```
3
SACTIV E1,0 E1,-1 !(!E0,1 !T0,1,OBSTA !T0,1,ACTIV) !(!
    T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD !T0,-1,SUPER
    !T0,-1,PERFE !T0,-1,SECYC !T0,-1,NWCYC !T0,-1,SWCYC
     !T0,-1,NECYC !T0,-1,WCYCL !T0,-1,ECYCL !T0,-1,
    SCYCL !T0,-1,NCYCL) !(T0,-1,STOPA T2,0,HEADD T2,-1,
    ACTIV E1,-1) !(T2,0,HEADD E1,-2 E2,-1 T2,-2,ACTIV
    (!T3,-1,STOPA !T3,-1,PIVOT))
P1,-1 A1111

%-------------------------------
%%%%%%%%% Deactivation of active
%-------------------------------

Deactivation to stopped
1
SACTIV
SSTOPA

%-------------------------------
%%%%%%%%% Stopeed activation
%-------------------------------

Activation by stopped N S E W
1
SSTOPA !(!(E0,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,1,STOPA !
    T1,1,PIVOT !T1,1,HEADD !T1,1,PERFE !T1,1,SECYC !T1
    ,1,NWCYC !T1,1,SWCYC !T1,1,NECYC !T1,1,WCYCL !T1,1,
    ECYCL !T1,1,SCYCL !T1,1,NCYCL) !(!T1,0,STOPA !T1,0,
    PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,SECYC !T1,0,
    NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,
    ECYCL !T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1 !(!E1,0 !T1
    ,0,OBSTA) !(!T-1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,
    HEADD !T-1,-1,PERFE !T-1,-1,SECYC !T-1,-1,NWCYC !T
    -1,-1,SWCYC !T-1,-1,NECYC !T-1,-1,WCYCL !T-1,-1,
    ECYCL !T-1,-1,SCYCL !T-1,-1,NCYCL) !(!T-1,0,STOPA !
    T-1,0,PIVOT !T-1,0,HEADD !T-1,0,PERFE !T-1,0,SECYC
    !T-1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL
     !T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL)) !(E1,0
    !(!E0,1 !T0,1,OBSTA) !(!T1,-1,STOPA !T1,-1,PIVOT !
    T1,-1,HEADD !T1,-1,PERFE !T1,-1,SECYC !T1,-1,NWCYC
    !T1,-1,SWCYC !T1,-1,NECYC !T1,-1,WCYCL !T1,-1,ECYCL
     !T1,-1,SCYCL !T1,-1,NCYCL) !(!T0,-1,STOPA !T0,-1,
    PIVOT !T0,-1,HEADD !T0,-1,PERFE !T0,-1,SECYC !T0
```

```
    ,-1,NWCYC !T0,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !
    T0,-1,ECYCL !T0,-1,SCYCL !T0,-1,NCYCL)) !(E-1,0 !(!
    E0,-1 !T0,-1,OBSTA) !(!T-1,1,STOPA !T-1,1,PIVOT !T
    -1,1,HEADD !T-1,1,PERFE !T-1,1,SECYC !T-1,1,NWCYC !
    T-1,1,SWCYC !T-1,1,NECYC !T-1,1,WCYCL !T-1,1,ECYCL
    !T-1,1,SCYCL !T-1,1,NCYCL) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC !T0,1,
    NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,
    ECYCL !T0,1,SCYCL !T0,1,NCYCL))) !T-1,1,ACTIV !T0
    ,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0
    ,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV !T-1,1,SUPER !
    T0,1,SUPER !T1,1,SUPER !T1,0,SUPER !T1,-1,SUPER !T0
    ,-1,SUPER !T-1,-1,SUPER !T-1,0,SUPER !(!(!T-1,0,
    STOPA !T-1,0,PIVOT) E1,0 !(!T0,-1,STOPA !T0,-1,
    PIVOT) !(!T1,-1,STOPA !T1,-1,PIVOT !T1,-1,HEADD)
    !(!T-1,-1,OBSTA !E-1,-1)) !(!(!T1,0,STOPA !T1,0,
    PIVOT) E-1,0 !(!T0,1,STOPA !T0,1,PIVOT) !(!T-1,1,
    STOPA !T-1,1,PIVOT !T-1,1,HEADD) !(!T1,1,OBSTA !E1
    ,1)) !(!(!T0,1,STOPA !T0,1,PIVOT) E0,-1 !(!T-1,0,
    STOPA !T-1,0,PIVOT) !(!T-1,-1,STOPA !T-1,-1,PIVOT !
    T-1,-1,HEADD) !(!T-1,1,OBSTA !E-1,1)) !(!(!T0,-1,
    STOPA !T0,-1,PIVOT) E0,1 !(!T1,0,STOPA !T1,0,PIVOT)
     !(!T1,1,STOPA !T1,1,PIVOT !T1,1,HEADD) !(!T1,-1,
    OBSTA !E1,-1))
SACTIV

Activation by stopped NE SW NW SE
1
SSTOPA !(!(E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,
    SECYC !T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,
    WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1
     E-1,-1 !(!E1,0 !T1,0,OBSTA) !(!T-1,0,STOPA !T-1,0,
    PIVOT !T-1,0,HEADD !T-1,0,PERFE !T-1,0,SECYC !T
    -1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL !
    T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL)) !(E-1,0 E
    -1,1 !(!E0,-1 !T0,-1,OBSTA) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC !T0,1,
    NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,
    ECYCL !T0,1,SCYCL !T0,1,NCYCL)) !(E1,0 E1,-1 !(!E0
    ,1 !T0,1,OBSTA) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,
    HEADD !T0,-1,PERFE !T0,-1,SECYC !T0,-1,NWCYC !T0
    ,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !T0,-1,ECYCL !
    T0,-1,SCYCL !T0,-1,NCYCL))) !T-1,1,ACTIV !T0,1,
```

```
    ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,
    ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV !T-1,1,SUPER !T0
    ,1,SUPER !T1,1,SUPER !T1,0,SUPER !T1,-1,SUPER !T0
    ,-1,SUPER !T-1,-1,SUPER !T-1,0,SUPER !(!(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD) !(!T1,1,OBSTA !E1,1) !(!T0,-1,
    OBSTA !E0,-1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T-1,0,
    STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,1,STOPA !T0
    ,1,PIVOT !T0,1,HEADD) !(!T-1,1,OBSTA !E-1,1) !(!T0
    ,-1,OBSTA !E0,-1) !(!T1,0,OBSTA !E1,0)) !(!(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,-1,STOPA !T0
    ,-1,PIVOT !T0,-1,HEADD) !(!T1,-1,OBSTA !E1,-1) !(!
    T0,1,OBSTA !E0,1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T
    -1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,-1,
    STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!T-1,-1,OBSTA !E
    -1,-1) !(!T0,1,OBSTA !E0,1) !(!T1,0,OBSTA !E1,0))
    !(!(!E-2,0 !T-2,0,ACTIV) T-1,0,PIVOT) !(!(!E2,0 !T2
    ,0,ACTIV) T1,0,PIVOT) !(!(!E0,2 !T0,2,ACTIV) T0,1,
    PIVOT) !(!(!E0,-2 !T0,-2,ACTIV) T0,-1,PIVOT) !(E0,1
     E1,0 T1,1,STOPA !(!T2,1,STOPA !T2,1,PIVOT) T2,0,
    ACTIV) !(E0,-1 E-1,0 T-1,-1,STOPA !(!T-2,-1,STOPA !
    T-2,-1,PIVOT) T-2,0,ACTIV) !(E1,0 E0,-1 T1,-1,STOPA
     !(!T1,-2,STOPA !T1,-2,PIVOT) T0,-2,ACTIV) !(E-1,0
    E0,1 T-1,1,STOPA !(!T-1,2,STOPA !T-1,2,PIVOT) T0,2,
    ACTIV)
SACTIV

%-------------------------------
%%%%%%%%% Pivot detection
%-------------------------------

Detect pivot
2
!(!SSTOPA !SACTIV) !(!(T1,0,STOPA !(!T1,1,STOPA !T1,1,
    PIVOT !E1,1) !(!T0,1,STOPA !T0,1,PIVOT !E0,1) !(!T
    -1,1,STOPA !T-1,1,PIVOT) T-1,0,HEADD) !(T-1,0,STOPA
     !(!T-1,-1,STOPA !T-1,-1,PIVOT !E-1,-1) !(!T0,-1,
    STOPA !T0,-1,PIVOT !E0,-1) !(!T1,-1,STOPA !T1,-1,
    PIVOT) T1,0,HEADD) !(T0,-1,STOPA !(!T1,-1,STOPA !T1
    ,-1,PIVOT !E1,-1) !(!T1,0,STOPA !T1,0,PIVOT !E1,0)
    !(!T1,1,STOPA !T1,1,PIVOT) T0,1,HEADD) !(T0,1,STOPA
     !(!T-1,1,STOPA !T-1,1,PIVOT !E-1,1) !(!T-1,0,STOPA
     !T-1,0,PIVOT !E-1,0) !(!T-1,-1,STOPA !T-1,-1,PIVOT
    ) T0,-1,HEADD))
```

```
SPIVOT

extra pivot detection oriented to superpivot
2
SSTOPA !(!(E-1,0 !(!T-1,1,STOPA !E-1,1) T0,1,STOPA T0
    ,-1,STOPA !(!T1,1,STOPA !T1,1,PIVOT) !(!T1,-1,STOPA
     !T1,-1,PIVOT) !(!T1,0,STOPA !T1,0,PIVOT) T-1,-1,
    OBSTA) !(E1,0 !(!T1,-1,STOPA !E1,-1) T0,-1,STOPA T0
    ,1,STOPA !(!T-1,-1,STOPA !T-1,-1,PIVOT) !(!T-1,1,
    STOPA !T-1,1,PIVOT) !(!T-1,0,STOPA !T-1,0,PIVOT) T1
    ,1,OBSTA) !(E0,1 !(!T1,1,STOPA !E1,1) T-1,0,STOPA
    T1,0,STOPA !(!T-1,-1,STOPA !T-1,-1,PIVOT) !(!T0,-1,
    STOPA !T0,-1,PIVOT) !(!T1,-1,STOPA !T1,-1,PIVOT) T
    -1,1,OBSTA) !(E0,-1 !(!T-1,-1,STOPA !E-1,-1) T1,0,
    STOPA T-1,0,STOPA !(!T1,1,STOPA !T1,1,PIVOT) !(!T0
    ,1,STOPA !T0,1,PIVOT) !(!T-1,1,STOPA !T-1,1,PIVOT)
    T1,-1,OBSTA))
SPIVOT

%-------------------------
%%%%%%%%% Pivot activation
%-------------------------

Activation by pivot N S E W
2
SPIVOT !(!(E0,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,1,STOPA !
    T1,1,PIVOT !T1,1,HEADD !T1,1,PERFE !T1,1,SECYC !T1
    ,1,NWCYC !T1,1,SWCYC !T1,1,NECYC !T1,1,WCYCL !T1,1,
    ECYCL !T1,1,SCYCL !T1,1,NCYCL) !(!T1,0,STOPA !T1,0,
    PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,SECYC !T1,0,
    NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,WCYCL !T1,0,
    ECYCL !T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1 !(!E1,0 !T1
    ,0,OBSTA) !(!T-1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,
    HEADD !T-1,-1,PERFE !T-1,-1,SECYC !T-1,-1,NWCYC !T
    -1,-1,SWCYC !T-1,-1,NECYC !T-1,-1,WCYCL !T-1,-1,
    ECYCL !T-1,-1,SCYCL !T-1,-1,NCYCL) !(!T-1,0,STOPA !
    T-1,0,PIVOT !T-1,0,HEADD !T-1,0,PERFE !T-1,0,SECYC
    !T-1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL
     !T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL)) !(E1,0
    !(!E0,1 !T0,1,OBSTA) !(!T1,-1,STOPA !T1,-1,PIVOT !
    T1,-1,HEADD !T1,-1,PERFE !T1,-1,SECYC !T1,-1,NWCYC
    !T1,-1,SWCYC !T1,-1,NECYC !T1,-1,WCYCL !T1,-1,ECYCL
     !T1,-1,SCYCL !T1,-1,NCYCL) !(!T0,-1,STOPA !T0,-1,
    PIVOT !T0,-1,HEADD !T0,-1,PERFE !T0,-1,SECYC !T0
```

```
    ,-1,NWCYC !T0,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !
    T0,-1,ECYCL !T0,-1,SCYCL !T0,-1,NCYCL)) !(E-1,0 !(!
    E0,-1 !T0,-1,OBSTA) !(!T-1,1,STOPA !T-1,1,PIVOT !T
    -1,1,HEADD !T-1,1,PERFE !T-1,1,SECYC !T-1,1,NWCYC !
    T-1,1,SWCYC !T-1,1,NECYC !T-1,1,WCYCL !T-1,1,ECYCL
    !T-1,1,SCYCL !T-1,1,NCYCL) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC !T0,1,
    NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,
    ECYCL !T0,1,SCYCL !T0,1,NCYCL))) !T-1,1,ACTIV !T0
    ,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0
    ,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV !T-1,1,SUPER !
    T0,1,SUPER !T1,1,SUPER !T1,0,SUPER !T1,-1,SUPER !T0
    ,-1,SUPER !T-1,-1,SUPER !T-1,0,SUPER !(!(!T-1,0,
    STOPA !T-1,0,PIVOT) E1,0 !(!T0,-1,STOPA !T0,-1,
    PIVOT) !(!T1,-1,STOPA !T1,-1,PIVOT !T1,-1,HEADD)
    !(!T-1,-1,OBSTA !E-1,-1)) !(!(!T1,0,STOPA !T1,0,
    PIVOT) E-1,0 !(!T0,1,STOPA !T0,1,PIVOT) !(!T-1,1,
    STOPA !T-1,1,PIVOT !T-1,1,HEADD) !(!T1,1,OBSTA !E1
    ,1)) !(!(!T0,1,STOPA !T0,1,PIVOT) E0,-1 !(!T-1,0,
    STOPA !T-1,0,PIVOT) !(!T-1,-1,STOPA !T-1,-1,PIVOT !
    T-1,-1,HEADD) !(!T-1,1,OBSTA !E-1,1)) !(!(!T0,-1,
    STOPA !T0,-1,PIVOT) E0,1 !(!T1,0,STOPA !T1,0,PIVOT)
     !(!T1,1,STOPA !T1,1,PIVOT !T1,1,HEADD) !(!T1,-1,
    OBSTA !E1,-1))
SACTIV


Activation by pivot NE SW NW SE
2
SPIVOT !(!(E0,1 E1,1 !(!E-1,0 !T-1,0,OBSTA) !(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD !T1,0,PERFE !T1,0,
    SECYC !T1,0,NWCYC !T1,0,SWCYC !T1,0,NECYC !T1,0,
    WCYCL !T1,0,ECYCL !T1,0,SCYCL !T1,0,NCYCL)) !(E0,-1
     E-1,-1 !(!E1,0 !T1,0,OBSTA) !(!T-1,0,STOPA !T-1,0,
    PIVOT !T-1,0,HEADD !T-1,0,PERFE !T-1,0,SECYC !T
    -1,0,NWCYC !T-1,0,SWCYC !T-1,0,NECYC !T-1,0,WCYCL !
    T-1,0,ECYCL !T-1,0,SCYCL !T-1,0,NCYCL)) !(E-1,0 E
    -1,1 !(!E0,-1 !T0,-1,OBSTA) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD !T0,1,PERFE !T0,1,SECYC !T0,1,
    NWCYC !T0,1,SWCYC !T0,1,NECYC !T0,1,WCYCL !T0,1,
    ECYCL !T0,1,SCYCL !T0,1,NCYCL)) !(E1,0 E1,-1 !(!E0
    ,1 !T0,1,OBSTA) !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,
    HEADD !T0,-1,PERFE !T0,-1,SECYC !T0,-1,NWCYC !T0
    ,-1,SWCYC !T0,-1,NECYC !T0,-1,WCYCL !T0,-1,ECYCL !
```

```
    T0,-1,SCYCL !T0,-1,NCYCL))) !T-1,1,ACTIV !T0,1,
    ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,
    ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV !T-1,1,SUPER !T0
    ,1,SUPER !T1,1,SUPER !T1,0,SUPER !T1,-1,SUPER !T0
    ,-1,SUPER !T-1,-1,SUPER !T-1,0,SUPER !(!(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,1,STOPA !T0,1,
    PIVOT !T0,1,HEADD) !(!T1,1,OBSTA !E1,1) !(!T0,-1,
    OBSTA !E0,-1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T-1,0,
    STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,1,STOPA !T0
    ,1,PIVOT !T0,1,HEADD) !(!T-1,1,OBSTA !E-1,1) !(!T0
    ,-1,OBSTA !E0,-1) !(!T1,0,OBSTA !E1,0)) !(!(!T1,0,
    STOPA !T1,0,PIVOT !T1,0,HEADD) !(!T0,-1,STOPA !T0
    ,-1,PIVOT !T0,-1,HEADD) !(!T1,-1,OBSTA !E1,-1) !(!
    T0,1,OBSTA !E0,1) !(!T-1,0,OBSTA !E-1,0)) !(!(!T
    -1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD) !(!T0,-1,
    STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!T-1,-1,OBSTA !E
    -1,-1) !(!T0,1,OBSTA !E0,1) !(!T1,0,OBSTA !E1,0))
    !(E0,1 E1,0 T1,1,STOPA !(!T2,1,STOPA !T2,1,PIVOT)
    T2,0,ACTIV) !(E0,-1 E-1,0 T-1,-1,STOPA !(!T-2,-1,
    STOPA !T-2,-1,PIVOT) T-2,0,ACTIV) !(E1,0 E0,-1 T1
    ,-1,STOPA !(!T1,-2,STOPA !T1,-2,PIVOT) T0,-2,ACTIV)
     !(E-1,0 E0,1 T-1,1,STOPA !(!T-1,2,STOPA !T-1,2,
    PIVOT) T0,2,ACTIV)
SACTIV

%-------------------------------
%%%%%%%%% Pivot deactivation
%-------------------------------

Pivot deactivation
1
SPIVOT !(!(!(!T1,0,STOPA !T1,0,HEADD !T1,0,PIVOT) !(!E
    -1,0 !T-1,0,OBSTA) (!T0,-1,STOPA !T0,-1,HEADD !T0
    ,-1,PIVOT) !(!E0,1 !T0,1,OBSTA) !(!E1,-1 !T1,-1,
    OBSTA)) !(!(!T-1,0,STOPA !T-1,0,HEADD !T-1,0,PIVOT)
     !(!E1,0 !T1,0,OBSTA) !(!T0,1,STOPA !T0,1,HEADD !T0
    ,1,PIVOT) !(!E0,-1 !T0,-1,OBSTA) !(!E-1,1 !T-1,1,
    OBSTA)) !(!(!T1,0,STOPA !T1,0,HEADD !T1,0,PIVOT)
    !(!E-1,0 !T-1,0,OBSTA) !(!T0,1,STOPA !T0,1,HEADD !
    T0,1,PIVOT) !(!E0,-1 !T0,-1,OBSTA) !(!E1,1 !T0,-1,
    OBSTA)) !(!(!T-1,0,STOPA !T-1,0,HEADD !T-1,0,PIVOT)
     !(!E1,0 !T1,0,OBSTA) !(!T0,-1,STOPA !T0,-1,HEADD !
    T0,-1,PIVOT) !(!E0,1 !T0,1,OBSTA) !(!E-1,-1 !T
    -1,-1,OBSTA)))
```

```
SSTOPA

%-------------------------------
%%%%%%%%%% Head transmission
%-------------------------------

Head transmission
2
SACTIV !(!T-1,0,HEADD !T0,1,HEADD !T1,0,HEADD !T0,-1,
    HEADD) !T1,1,HEADD !T1,-1,HEADD !T-1,1,HEADD !T
    -1,-1,HEADD !(!T-1,1,OBSTA !T0,1,OBSTA !T1,1,OBSTA
    !T1,0,OBSTA !T1,-1,OBSTA !T0,-1,OBSTA !T-1,-1,OBSTA
     !T-1,0,OBSTA)
SHEADD

%-------------------------------
%%%%%%%%%% Head deactivation
%-------------------------------

No longer head
2
SHEADD !(!(T0,1,HEADD T1,0,OBSTA) !(T-1,0,HEADD T0,1,
    OBSTA) !(T0,-1,HEADD T-1,0,OBSTA) !(T1,0,HEADD T0
    ,-1,OBSTA) !(T0,1,HEADD T1,1,OBSTA) !(T-1,0,HEADD T
    -1,1,OBSTA) !(T0,-1,HEADD T-1,-1,OBSTA) !(T1,0,
    HEADD T1,-1,OBSTA) !(T0,1,HEADD E-1,0 E1,0 E1,1 E
    -1,1 !(!T1,-1,OBSTA !T-1,-1,OBSTA)) !(T0,-1,HEADD E
    -1,0 E1,0 E-1,-1 E1,-1 !(!T-1,1,OBSTA !T1,1,OBSTA))
     !(T1,0,HEADD E0,1 E0,-1 E1,1 E1,-1 !(!T-1,1,OBSTA
    !T-1,-1,OBSTA)) !(T-1,0,HEADD E0,-1 E0,1 E-1,-1 E
    -1,1 !(!T1,-1,OBSTA !T1,1,OBSTA)) !(T0,1,HEADD T
    -1,1,STOPA E1,1 E1,0 E1,-1 !(!E-1,0 !T-1,0,ACTIV)
    !(!E-1,-1 !T-1,-1,ACTIV)) !(T0,-1,HEADD T1,-1,STOPA
     E-1,-1 E-1,0 E-1,1 !(!E1,0 !T1,0,ACTIV) !(!E1,1 !
    T1,1,ACTIV)) !(T1,0,HEADD E1,-1 T1,1,STOPA E0,-1 E
    -1,-1 T-1,0,STOPA !(!T0,1,ACTIV !E0,1) !(!T-1,1,
    ACTIV !E-1,1)) !(T-1,0,HEADD E-1,1 T-1,-1,STOPA E0
    ,1 E1,1 T1,0,STOPA !(!T0,-1,ACTIV !E0,-1) !(!T1,-1,
    ACTIV !E1,-1)))
SSTOPA

%-------------------------------
%%%%%%%%%% Head readjustment
%-------------------------------
```

```
Head readjustment
2
SSTOPA !(!(T0,-1,HEADD !(!T0,1,STOPA !T0,1,OBSTA !E0
    ,1) T1,0,OBSTA) !(T0,1,HEADD !(!T0,-1,STOPA !T0,-1,
    OBSTA E0,-1) T-1,0,OBSTA) !(T-1,0,HEADD !(!T1,0,
    STOPA !T1,0,OBSTA !E1,0) T0,-1,OBSTA) !(T1,0,HEADD
    !(!T-1,0,STOPA !T-1,0,OBSTA !E-1,0) T0,1,OBSTA)
    !(!(!T1,0,STOPA !T1,0,OBSTA !E1,0) T0,-1,HEADD T1
    ,-1,OBSTA) !(!(!T-1,0,STOPA !T-1,0,OBSTA !E-1,0) T0
    ,1,HEADD T-1,1,OBSTA) !(!(!T0,-1,STOPA !T0,-1,OBSTA
     !E0,-1) T-1,0,HEADD T-1,-1,OBSTA) !(!(!T0,1,STOPA
    !T0,1,OBSTA !E0,1) T1,0,HEADD T1,1,OBSTA) !(T0,-1,
    HEADD E-1,0 E-1,-1 E1,0 E1,-1 E1,1 T0,1,STOPA T
    -1,1,STOPA T0,-2,STOPA) !(T0,1,HEADD E1,0 E1,1 E
    -1,0 E-1,1 E-1,-1 T0,-1,STOPA T1,-1,STOPA T0,2,
    STOPA) !(T-1,0,HEADD T1,0,STOPA E-1,1 E0,1 T1,1,
    STOPA E-1,-1 E0,-1 E1,-1 T1,1,STOPA T-2,0,STOPA) !(
    T-1,0,HEADD T-1,0,STOPA E1,-1 E0,-1 T-1,-1,STOPA E1
    ,1 E0,1 E-1,1 T-1,-1,STOPA T2,0,STOPA) !(T-1,0,
    STOPA T0,-1,HEADD T0,1,STOPA !(!E-1,1 !T-1,1,STOPA)
     T1,1,OBSTA E1,0 E1,-1 E1,-2 T0,-2,STOPA !(!T-1,-1,
    ACTIV !E-1,-1) !(!T-1,-2,ACTIV !E-1,-2)) !(T1,0,
    STOPA T0,1,HEADD T0,-1,STOPA !(!E1,-1 !T1,-1,STOPA)
     T-1,-1,OBSTA E-1,0 E-1,1 E-1,2 T0,2,STOPA !(!T1,1,
    ACTIV !E1,1) !(!T1,2,ACTIV !E1,2)) !(T-1,0,STOPA T
    -1,1,OBSTA !(!E-1,-1 !T-1,-1,STOPA) T0,-1,STOPA E0
    ,1 E1,1 T1,0,HEADD E2,1 T2,0,STOPA !(!T1,-1,ACTIV !
    E1,-1) !(!T2,-1,ACTIV !E2,-1)) !(T1,0,STOPA T1,-1,
    OBSTA !(!E1,1 !T1,1,STOPA) T0,1,STOPA E0,-1 E-1,-1
     T-1,0,HEADD E-2,-1 T-2,0,STOPA !(!T-1,1,ACTIV !E
    -1,1) !(!T-2,1,ACTIV !E-2,1)))
SHEADD

%-------------------------------
%%%%%%%%% Superactive movement
%-------------------------------

Superactive N movement
2
SSUPER E0,1 !(!T1,1,STOPA !T1,1,PIVOT !T1,1,HEADD) !(!
    T1,0,STOPA !T1,0,PIVOT) !T0,1,ACTIV !T1,1,ACTIV !T1
    ,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T
    -1,0,ACTIV
```

```
P0,1 A1111 SACTIV

Superactive S movement
2
SSUPER E0,-1 !(!T-1,-1,STOPA !T-1,-1,PIVOT !T-1,-1,
    HEADD) !(!T-1,0,STOPA !T-1,0,PIVOT) !T0,1,ACTIV !T1
    ,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T
    -1,-1,ACTIV !T-1,0,ACTIV
P0,-1 A1111 SACTIV

Superactive E movement
2
SSUPER E1,0 !(!T1,-1,STOPA !T1,-1,PIVOT !T1,-1,HEADD)
    !(!T0,-1,STOPA !T0,-1,PIVOT) !T0,1,ACTIV !T1,1,
    ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T
    -1,-1,ACTIV !T-1,0,ACTIV
P1,0 A1111 SACTIV

Superactive W movement
2
SSUPER E-1,0 !(!T-1,1,STOPA !T-1,1,PIVOT !T-1,1,HEADD)
     !(!T0,1,STOPA !T0,1,PIVOT) !T0,1,ACTIV !T1,1,ACTIV
     !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,
    ACTIV !T-1,0,ACTIV
P-1,0 A1111 SACTIV

Superactive NE movement
2
SSUPER E0,1 E1,1 !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD
    ) !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV
    !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P1,1 A1111 SACTIV

Superactive SW movement
2
SSUPER E0,-1 E-1,-1 !(!T-1,0,STOPA !T-1,0,PIVOT !T
    -1,0,HEADD) !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1
    ,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P-1,-1 A1111 SACTIV

Superactive SE movement
2
SSUPER E1,0 E1,-1 !(!T0,-1,STOPA !T0,-1,PIVOT !T-1,0,
    HEADD) !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
```

```
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P1,-1 A1111 SACTIV

Superactive NW movement
2
SSUPER E-1,0 E-1,1 !(!T0,1,STOPA !T0,1,PIVOT !T0,1,
    HEADD) !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P-1,1 A1111 SACTIV

%-----------------------------
%%%%%%%%% Superactive detection
%-----------------------------

Detect superactive
2
SPIVOT !(!(E0,-1 T1,0,STOPA (!T0,1,OBSTA !T0,1,ACTIV !
    E0,1) !(!T-1,0,STOPA !T-1,0,PIVOT !T-1,0,HEADD)
    !(!(T1,-1,OBSTA) !(E1,-1 T2,0,STOPA T2,-1,OBSTA)))
    !(E0,1 T-1,0,STOPA (!T0,-1,OBSTA !T0,-1,ACTIV !E0
    ,-1) !(!T1,0,STOPA !T1,0,PIVOT !T1,0,HEADD) !(!(T
    -1,1,OBSTA) !(E-1,1 T-2,0,STOPA T-2,1,OBSTA))) !(E1
    ,0 T0,1,STOPA (!T-1,0,OBSTA !T-1,0,ACTIV !E-1,0)
    !(!T0,-1,STOPA !T0,-1,PIVOT !T0,-1,HEADD) !(!(T1,1,
    OBSTA) !(E1,1 T0,2,STOPA T1,2,OBSTA))) !(E-1,0 T0
    ,-1,STOPA (!T1,0,OBSTA !T1,0,ACTIV !E1,0) !(!T0,1,
    STOPA !T0,1,PIVOT !T0,1,HEADD) !(!(T-1,-1,OBSTA) !(
    E-1,-1 T0,-2,STOPA T-1,-2,OBSTA)))) !T-1,1,ACTIV !
    T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0
    ,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV !T0,2,ACTIV !
    T1,2,ACTIV !T2,2,ACTIV !T2,1,ACTIV !T2,0,ACTIV !T2
    ,-1,ACTIV !T2,-2,ACTIV !T1,-2,ACTIV !T0,-2,ACTIV !T
    -1,-2,ACTIV !T-2,-2,ACTIV !T-2,-1,ACTIV !T-2,0,
    ACTIV !T-2,1,ACTIV !T-2,2,ACTIV !T-1,2,ACTIV
SSUPER


%-----------------------------
%%%%%%%%% Perfect cycles
%-----------------------------

Detect perfect cycle
1
!(!SSTOPA !SPIVOT) !(!(E0,1 E0,-1 E1,1 !(!E-1,-1 !(!(!
```

```
    T-1,-1,PIVOT !T-1,-1,STOPA) !(!T-2,-1,OBSTA !T-2,0,
    OBSTA))) T1,0,HEADD !(!T-1,0,STOPA !T-1,0,PIVOT))
    !(E0,-1 E0,1 E-1,-1 !(!E1,1 !(!(!T1,1,PIVOT !T1,1,
    STOPA) !(!T2,1,OBSTA !T2,0,OBSTA))) T-1,0,HEADD !(!
    T1,0,STOPA !T1,0,PIVOT)) !(E1,0 E1,-1 E-1,0 !(!E
    -1,1 !(!(!T-1,1,PIVOY !T-1,1,STOPA) !(!T-1,2,OBSTA
    !T0,2,OBSTA))) T0,-1,HEADD !(!T0,1,STOPA !T0,1,
    PIVOT)) !(E-1,0 E-1,1 E1,0 !(!E1,-1 !(!(!T1,-1,
    PIVOY !T1,-1,STOPA) !(!T1,-2,OBSTA !T0,-2,OBSTA)))
    T0,1,HEADD !(!T0,-1,STOPA !T0,-1,PIVOT)) !(E0,1 E1
    ,1 E1,-1 !(!T-1,0,OBSTA !T-1,1,OBSTA) T1,0,HEADD
    !(!T0,-1,STOPA !T0,-1,PIVOT)) !(E0,-1 E-1,-1 E-1,1
    !(!T1,0,OBSTA !T1,-1,OBSTA) T-1,0,HEADD !(!T0,1,
    STOPA !T0,1,PIVOT)) !(E1,0 E1,-1 E-1,-1 !(!T0,1,
    OBSTA !T1,1,OBSTA) T0,-1,HEADD !(!T0,1,STOPA !T0,1,
    PIVOT)) !(E-1,0 E-1,1 E1,1 !(!T0,-1,OBSTA !T-1,-1,
    OBSTA) T0,1,HEADD !(!T0,-1,STOPA !T0,-1,PIVOT))) !T
    -1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1
    ,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
SPERFE

Perfect NE movement
1
SPERFE E0,1 E1,1 T1,0,HEADD !T0,1,ACTIV !T1,1,ACTIV !
    T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV
    !T-1,0,ACTIV
P1,1 A1111 SACTIV

Perfect SE movement
1
SPERFE E1,0 E1,-1 T0,-1,HEADD !T0,1,ACTIV !T1,1,ACTIV
    !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV
     !T-1,0,ACTIV
P1,-1 A1111 SACTIV

Perfect SW movement
1
SPERFE E0,-1 E-1,-1 T-1,0,HEADD !T0,1,ACTIV !T1,1,
    ACTIV !T1,0,ACTIV !T1,-1,ACTIV !T0,-1,ACTIV !T
    -1,-1,ACTIV !T-1,0,ACTIV
P-1,-1 A1111 SACTIV

Perfect NW movement
1
```

```
SPERFE E-1,0 E-1,1 T0,1 ,HEADD !T0,1 ,ACTIV !T1,1 ,ACTIV
    !T1,0 ,ACTIV !T1,-1 ,ACTIV !T0,-1 ,ACTIV !T-1,-1 ,ACTIV
     !T-1,0 ,ACTIV
P-1,1 A1111 SACTIV

Perfect deactivation
1
SPERFE !T-1,0 ,HEADD !T0,1 ,HEADD !T0,-1 ,HEADD !T1,0 ,
   HEADD
SACTIV


%-------------------------------
%%%%%%%%% T structure
%-------------------------------

N T structure
1
!(!SPIVOT !SSTOPA) !(!(!(!T-1,0 ,STOPA !T-1,0 ,PIVOT) E0
    ,1 E0,-1 !(!T1,1 ,STOPA !T1,1 ,HEADD) T1,0 ,STOPA !(!
    T1,-1 ,STOPA !T1,-1 ,PIVOT) T2,1 ,OBSTA !(!T2,0 ,STOPA
    !T2,0 ,PIVOT !T2,0 ,OBSTA !E2,0) !(!T2,-1 ,STOPA !T2
    ,-1 ,PIVOT !T2,-1 ,OBSTA !E2,-1)) !(!(!T-1,0 ,STOPA !T
    -1,0 ,PIVOT) E0,1 E0,-1 E1,-1 T1,0 ,STOPA !(!T1,1 ,
    STOPA !T1,1 ,HEADD) T2,1 ,OBSTA !(!T2,0 ,STOPA !T2,0 ,
    PIVOT) !(!T2,-1 ,STOPA !T2,-1 ,PIVOT !E2,-1)) !(T
    -1,0 ,OBSTA E0,1 !(!T0,-1 ,STOPA !T0,-1 ,PIVOT) !(!T1
    ,1 ,STOPA !T1,1 ,HEADD) T2,1 ,OBSTA T1,0 ,STOPA E1,-1
    T2,0 ,STOPA !(!E2,-1 !T2,-1 ,STOPA !T2,-1 ,PIVOT))) !
    T0,1 ,ACTIV !T1,1 ,ACTIV !T1,0 ,ACTIV !T1,-1 ,ACTIV !T0
    ,-1 ,ACTIV !T-1,-1 ,ACTIV !T-1,0 ,ACTIV
SNCYCL

NCYCL movement
1
SNCYCL !T0,1 ,ACTIV !T1,1 ,ACTIV !T1,0 ,ACTIV !T1,-1 ,
   ACTIV !T0,-1 ,ACTIV !T-1,-1 ,ACTIV !T-1,0 ,ACTIV
P0,1 A1111 SACTIV

S T structure
1
!(!SPIVOT !SSTOPA) !(!(!(!T1,0 ,STOPA !T1,0 ,PIVOT) E0
    ,-1 E0,1 !(!T-1,-1 ,STOPA !T-1,-1 ,HEADD) T-1,0 ,STOPA
     !(!T-1,1 ,STOPA !T-1,1 ,PIVOT) T-2,-1 ,OBSTA !(!T
```

```
    −2,0,STOPA !T−2,0,PIVOT !T−2,0,OBSTA !E−2,0) !(!T
    −2,1,STOPA !T−2,1,PIVOT !T−2,1,OBSTA !E−2,1)) !(!(!
    T1,0,STOPA !T1,0,PIVOT) E0,−1 E0,1 E−1,1 T−1,0,
    STOPA !(!T−1,−1,STOPA !T−1,−1,HEADD) T−2,−1,OBSTA
    !(!T−2,0,STOPA !T−2,0,PIVOT) !(!T−2,1,STOPA !T−2,1,
    PIVOT !E−2,1)) !(T1,0,OBSTA E0,−1 !(!T0,1,STOPA !T0
    ,1,PIVOT) !(!T−1,−1,STOPA !T−1,−1,HEADD) T−2,−1,
    OBSTA T−1,0,STOPA E−1,1 T−2,0,STOPA !(!E−2,1 !T
    −2,1,STOPA !T−2,1,PIVOT))) !T0,−1,ACTIV !T−1,−1,
    ACTIV !T−1,0,ACTIV !T−1,1,ACTIV !T0,1,ACTIV !T1,1,
    ACTIV !T1,0,ACTIV
SSCYCL

SCYCL movement
1
SSCYCL !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,−1,
    ACTIV !T0,−1,ACTIV !T−1,−1,ACTIV !T−1,0,ACTIV
P0,−1 A1111 SACTIV

E T structure
1
!(!SPIVOT !SSTOPA) !(!(!(!T0,1,STOPA !T0,1,PIVOT) E
    −1,0 E1,0 !(!T1,−1,STOPA !T1,−1,HEADD) T0,−1,STOPA
    !(!T−1,−1,PIVOT !T−1,−1,STOPA) T1,−2,OBSTA !(!T0
    ,−2,STOPA !T0,−2,PIVOT !T0,−2,OBSTA !E0,−2) !(!T
    −1,−2,STOPA !T−1,−2,PIVOT !T−1,−2,OBSTA !E−1,−2))
    !(!(!T0,1,STOPA !T0,1,PIVOT) E−1,0 E1,0 !(!T1,−1,
    STOPA !T1,−1,HEADD) T0,−1,STOPA E−1,−1 !(!T0,−2,
    STOPA !T0,−2,PIVOT) !(!T−1,−2,STOPA !T−1,−2,PIVOT !
    E−1,−2)) !(T0,1,OBSTA !(!T−1,0,STOPA !T−1,0,PIVOT)
    E1,0 !(!T1,−1,STOPA !T1,−1,HEADD) T0,−1,STOPA E
    −1,−1 T0,−2,STOPA T1,−2,OBSTA !(!E−1,−2 !T−1,−2,
    STOPA !T−1,−2,PIVOT))) !T0,−1,ACTIV !T−1,−1,ACTIV !
    T−1,0,ACTIV !T−1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !
    T1,0,ACTIV
SECYCL

ECYCL movement
1
SECYCL !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,−1,
    ACTIV !T0,−1,ACTIV !T−1,−1,ACTIV !T−1,0,ACTIV
P1,0 A1111 SACTIV

W T structure
```

```
1
!(!SPIVOT !SSTOPA) !(!(!(!T0,-1,STOPA !T0,-1,PIVOT) E1
    ,0 E-1,0 !(!T-1,1,STOPA !T-1,1,HEADD) T0,1,STOPA
    !(!T1,1,PIVOT !T1,1,STOPA) T-1,2,OBSTA !(!T0,2,
    STOPA !T0,2,PIVOT !T0,2,OBSTA !E0,2) !(!T1,2,STOPA
    !T1,2,PIVOT !T1,2,OBSTA !E1,2)) !(!(!T0,-1,STOPA !
    T0,-1,PIVOT) E1,0 E-1,0 !(!T-1,1,STOPA !T-1,1,HEADD
    ) T0,1,STOPA E1,1 !(!T0,2,STOPA !T0,2,PIVOT) !(!T1
    ,2,STOPA !T1,2,PIVOT !E1,2)) !(T0,-1,OBSTA !(!T1,0,
    STOPA !T1,0,PIVOT) E-1,0 !(!T-1,1,STOPA !T-1,1,
    HEADD) T0,1,STOPA E1,1 T0,2,STOPA T-1,2,OBSTA !(!E1
    ,2 !T1,2,STOPA !T1,2,PIVOT))) !T0,-1,ACTIV !T-1,-1,
    ACTIV !T-1,0,ACTIV !T-1,1,ACTIV !T0,1,ACTIV !T1,1,
    ACTIV !T1,0,ACTIV
SWCYCL

WCYCL movement
1
SWCYCL !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P-1,0 A1111 SACTIV

NE T structure
1
!(!SPIVOT !SSTOPA) !(!T-1,0,STOPA !T-1,0,PIVOT) E0,1
    E0,-1 E1,1 !(!T1,0,STOPA !T1,0,PIVOT) !(!T1,-1,
    STOPA !T1,-1,PIVOT) !(!T2,1,STOPA !T2,1,PIVOT !E2,1
     !T2,1,HEADD) !(!T2,0,STOPA !T2,0,HEADD) !(!T2,-1,
    OBSTA !T2,-1,STOPA) !T0,-1,ACTIV !T-1,-1,ACTIV !T
    -1,0,ACTIV !T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1
    ,0,ACTIV
SNECYC

NECYC movement
1
SNECYC !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P1,1 A1111 SACTIV

SW T structure
1
!(!SPIVOT !SSTOPA) !(!T1,0,STOPA !T1,0,PIVOT) E0,-1 E0
    ,1 E-1,-1 !(!T-1,0,STOPA !T-1,0,PIVOT) !(!T-1,1,
    STOPA !T-1,1,PIVOT) !(!T-2,-1,STOPA !T-2,-1,PIVOT !
```

```
    E-2,-1 !T-2,-1,HEADD) !(!T-2,0,STOPA !T-2,0,HEADD)
    !(!T-2,1,OBSTA !T-2,1,STOPA) !T0,-1,ACTIV !T-1,-1,
    ACTIV !T-1,0,ACTIV !T-1,1,ACTIV !T0,1,ACTIV !T1,1,
    ACTIV !T1,0,ACTIV
SSWCYC

SWCYC movement
1
SSWCYC !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P-1,-1 A1111 SACTIV

NW T structure
1
!(!SPIVOT !SSTOPA) !(!T0,-1,STOPA !T0,-1,PIVOT) E-1,0
    E1,0 E-1,1 !(!T0,1,PIVOT !T0,1,STOPA) !(!T1,1,PIVOT
     !T1,1,STOPA) !(!T0,2,HEADD !T0,2,STOPA) !(!E-1,2 !
    T-1,2,STOPA !T-1,2,HEADD !T-1,2,PIVOT) !(!T1,2,
    OBSTA !T1,2,STOPA) !T0,-1,ACTIV !T-1,-1,ACTIV !T
    -1,0,ACTIV !T-1,1,ACTIV !T0,1,ACTIV !T1,1,ACTIV !T1
    ,0,ACTIV
SNWCYC

NWCYC movement
1
SNWCYC !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
P-1,1 A1111 SACTIV

SE T structure
1
!(!SPIVOT !SSTOPA) !(!T0,1,STOPA !T0,1,PIVOT) E1,0 E
    -1,0 E1,-1 !(!T0,-1,PIVOT !T0,-1,STOPA) !(!T-1,-1,
    PIVOT !T-1,-1,STOPA) !(!T0,-2,HEADD !T0,-2,STOPA)
    !(!E1,-2 !T1,-2,STOPA !T1,-2,HEADD !T1,-2,PIVOT)
    !(!T-1,-2,OBSTA !T-1,-2,STOPA) !T0,-1,ACTIV !T
    -1,-1,ACTIV !T-1,0,ACTIV !T-1,1,ACTIV !T0,1,ACTIV !
    T1,1,ACTIV !T1,0,ACTIV
SSECYC

SECYC movement
1
SSECYC !T0,1,ACTIV !T1,1,ACTIV !T1,0,ACTIV !T1,-1,
    ACTIV !T0,-1,ACTIV !T-1,-1,ACTIV !T-1,0,ACTIV
```

P1,−1 A1111 SACTIV