Algorithmic Transformations and Constructions for Programmable Matter and other Networked Systems

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy by

Matthew Connor

June 2023

Contents

Illustrations

List of Figures

List of Tables

Abstract

Programmable matter is a recently developed research area tasked with investigating the properties of systems made up of small, weak computing entities co-operating as components of a single coherent object to accomplish non-trivial tasks. A key characteristic of programmable matter systems is the dynamics, whether the computers in the system act independently or react to changes in the environment. In this thesis, we study active models of programmable matter, meaning that at control lies with the computing entities. We are interested in exploring the capabilities of these systems and defining which tasks can be solved by them. We investigate the rotation model, a model where the entities form a shape on a 2D grid and move by rotating around each other. Previous work introduced the RotC-Transformability problem, where shapes must be transformed without any of the entities disconnecting from the shape at any point. We solve this problem for the *nice* and *orthogonally convex* classes of shapes by providing algorithms to transform between shapes in the same class.

In Chapter 2, we define the rotation model, as well as some other preliminaries.

In Chapter 3, we present our nice shapes construction result by exploiting the concept of canonical shapes to transform a line into a nice shape, and then demonstrate that the transformation of nice shapes into any other nice shape through the canonical shape is possible. We exploit the concept of *seeds*, which are small shapes added to the main shape to aid construction. This is justified by the presence of shapes which cannot be transformed without external aid. We use the seed to construct a *robot*, a subset of the computing entities, to move the components of the shape efficiently.

In Chapter 4, we give our orthogonally convex shapes result by using the same method. However, the result is much more technically complex, with traversal results which were harder to establish due the larger space of local shape configurations which needed to be considered, including a larger number of edge cases. To make this work, we needed to make use of two types of canonical shapes, an intermediate one and a final one, which was necessary to ensure certain difficult shapes could be transformed without breaking connectivity. This class of shapes is richer in structure than nice shapes, and therefore the result moves closer to the goal of universal transformation, the ability to transform any shape into any other shape.

Acknowledgements

I am thankful to my supervisor, Othon. He has helped me enormously from the very beginning and without him I would not have accomplished as much as I have. He has given me help and advice almost every week, and his passion for his research and willingness to go the extra mile has been invaluable.

I am also thankful to Igor and Paul, who have helped by chipping in at certain points. I thank my academic advisors Leszek and Prudence for assessing my progress every year and giving helpful advice. I would like to thank the staff of the University of Liverpool for keeping the place running, especially during the pandemic, and for supporting me with funding and travel advice. I would also like to thank the Networks and Distributed Computing group for the introducing me to different topics with their weekly presentations.

Finally, I would like to thank my family for supporting me in my PhD journey.

Chapter 1

Introduction

1.1 Dynamic Systems of Programmable Entities

The theory of programmable matter is a research area within the theoretical computer science field investigating the algorithmic properties of models used to represent programmable entities, groups of identical objects, referred to as *nodes* throughout. They exist within an environment and are co-ordinated by one or more programs, computing decisions over a set of potential actions to decide which actions the objects should take. These objects in turn form a single coherent object, with the preservation of connectivity, the physical connections between nodes which keep the object coherent, one of the defining characteristics of programmable matter.

Research in this area is increasingly popular, motivated by the existence of systems which can be represented by programmable matter models. For example, within nature there are systems consisting of subcomponents capable of autonomous behaviour like the regeneration of tissue in biological organisms. The operations of biological processes such as these can be represented by algorithms within a programmable matter model. The algorithms and models developed with this method can be used to enable and expediate the development of efficient centralised or decentralised algorithms and programmable matter systems, the results of which can then be applied to existing technologies.

A key property of these systems is the control of the *dynamics*, or the changes which take place within the system. These changes are represented as a transition between states. A programmable matter system is *actively dynamic* if all change is the result of actions which the objects make through the execution of a predetermined algorithm. Examples of systems with active dynamics include reconfigurable robots, such as molecubes [1] which can move around and change the structure of the system. In addition, systems which rely on large collectives of identical robots have been developed, for example the Kilobot system [2] and the Robot Pebbles system [3]. Another interesting implementation is Millimotein [4], a system where programmable matter folds itself into arbitrary 3D shapes. The Catoms system [5–7] is a further implementation which

constructs 3D shapes by first creating a "scaffolding structure" as a basis for construction. A more recent variant is Datom, which relies on the robot deforming its shape to move [8, 9].

Programmable matter systems in turn have theoretical models to represent them. There are models for programmable matter which are actively dynamic, for example the SILBOT model [10], and the nubot model [11]. There is also extensive research into the amoebot model [12–15], where finite automata on a triangular lattice follow a distributed algorithm to achieve a desired goal, including a recent extension [16] to a circuit-based model and another extension [17] which introduced concurrency control. There is also research into the potential for fault tolerance in the model [18]. It is expected that applications in further domains such as molecular computers and self-repairing machines may become apparent in the long-term. Another model is the Crystalline robots model [19], where unit-cube atoms move by extending and contracting arms, which can attach to and detach from neighbours.

Programmable matter is closely related to *actively dynamic networks*. These are more general network structures which give the nodes within the network control over the dynamics, for example the ability to place and remove edges in the network graph. In a sense, programmable matter is the geometric equivalent of these kinds of networks, with a greater emphasis on the close proximity of the nodes. A recent paper [20] defines complexity measures for measuring the costs of creating and maintaining networks in this scenario, and provides algorithms both for reconfiguring the network and solving a general task on it. Applications, both theoretical and in systems, include peer-to-peer networks [21], wireless communication networks [22] and transportation networks [23]. There may be potential applications for a hybrid system, where nodes alternate between programmable matter and dynamic network models depending their proximity to each other.

Another related area of study is swarm robotics. These are systems for organising a group of robots which operate as a decentralised swarm, as opposed to acting as subcomponents of a coherent object. The result is more relaxed rules for movement which are independent of other robots, rather than relative to other robots as in programmable matter. There are models and algorithms for gathering [24, 25], deployment [26, 27], geometric pattern formation [28, 29] and connectivity preservation [30]. A hybridisation of these models with programmable matter would yield models of systems where nodes can alternate between the formation of a coherent object and more independent behaviour. A similar concept is that of programmable matter models which allow nodes to break connectivity, as in [31].

On the other hand, a programmable matter system is *passive* if the dynamics are external to the system, and therefore outside of its control. For example, the behaviour of a system which monitors woodland to detect fires depends on the state of the area

it is monitoring. Other examples include self-assembling programmable materials, such as the Abstract Tile Assembly Model [32–34] and self-assembling DNA molecules [35, 36], which form bonds based on the random interaction of components (see also a general survey in [37]). Another interesting example is slime molds, which are able to solve computational problems [38, 39] by foraging for food according to predetermined behaviour. Recent work has resulted in a system of particles which use DNA sequences to merge, with the properties of both the particles and the DNA sequences able to be customised to create unique materials [40].

In models of these systems, the environment imposes events on the system, and the system reacts according to a predetermined algorithm. More specifically, the events which occur are selected from a pool of potential events by a *scheduler*. There are various types of scheduler, for example the *uniform random* scheduler, which selects events uniformly at random, and the *adversarial* scheduler, which selects the event which creates the least progress towards a goal specified when the algorithm is defined. Such schedulers are almost always required to be *fair*, meaning that they select every possible event infinitely often, so that trivial loops do not prevent the algorithm from operating correctly. An example of a model representing the properties of passively dynamic programmable matter systems is the population protocols model of Angluin *et al.* [41, 42], where nodes are only capable of changing state in response to (typically pairwise) interactions with other nodes. Another is the model by Kuhn *et al.* [43] where nodes in a network have potential connections which are controlled by an adversarial scheduler. Michail *et al.* [44] extended the latter model to the case of possibly disconnected dynamic networks, in which connectivity is only guaranteed in a temporal sense. The Tile Automata model [45] is a recent model combining features of both cellular automata and the 2-Handed model of self-assembly. There is also the model by Emek and Uitto inspired by multicellular biological processes, where nodes must try to confine changes to their immediate neighbourhood [46]. There are several applications for passive systems, for example the modelling of chemical reaction networks [47, 48] and epidemics [49].

Some programmable matter systems are *hybrid* systems. In models of these systems, the nodes exist within an environment which imposes events on the system, however, the nodes have the ability to take actions which alter the environment, in turn altering the events it imposes in a kind of feedback loop. Nodes may even be able to create structures which the environment cannot control. An example of a hybrid system is the tile line system, where a line of passive tiles is maintained by an active robot [50].

Examples of hybrid models include the network constructors model introduced by Michail and Spirakis [51], an abstract model of distributed network construction where the network dynamicity is the same as in population protocols but now the finite-state entities can additionally activate and deactivate pairwise connections upon their

interactions. These connections in turn change the events which occur, as the outcome of events can differ depending on whether or not a connection exists between the nodes when an interaction occurs. This is naturally motivated by molecular interactions where, for example, proteins can bind to each other, forming structures and maintaining their stability despite the dynamicity of the solution in which they reside. This model is similar to the mediated population protocols model [52], which differs from the former model by having a constant-size set of states for connections, not just a binary for existence. This shift, from constant edge-states to binary edge-states, is due to the change in focus away from computation problems and towards network construction. Then Michail [53] studied a geometric variant of network constructors, in which the entities can only form geometrically constrained shapes in 2D or 3D space. Another interesting hybrid dynamic network model is the one by Gmyr *et al.* [54], in which the entities have partial control over the connections of an otherwise worst-case passively dynamic network, following the model of Kuhn *et al.* [43].

The foundations of programmable matter are based on work with algorithms which represent the capabilities of a generic programmable matter system. For example, Czyzowicz *et al.* [55] provide a bound on the speed that a generic active system can construct compact structures where the robot is a deterministic finite automaton. In addition, work on a generic model of programmable matter called "self-organising particle systems" has provided insights into fundamental properties such as compression of nodes in space [56], the election of "leader" nodes for breaking symmetry in algorithms [57], and other algorithms such as shape formation and shape recognition [58]. There is also work on universal coating, or covering the whole surface of a shape with nodes [59]. The impact of energy constraints on behaviour, inspired by natural systems, has also been explored [60]. Programmable matter systems have been shown to be able to mimic the operations of CAD programs [61]. On the side of passively dynamic models, work on DNA self-assembly and abstract tile assembly is inherently algorithmic, as the passive nature of these systems mean they follow an algorithm in response to events, which act as inputs, imposed by the environment. Population protocols have been shown to be able to compute semilinear predicates [62], and simulate a Turing machine via the use of a "phase clock", which propagates phases of a computation like an epidemic [63]. Population protocols usually assign an initial configuration to the population as part of the input. Self-stabilisation, or the ability of populations with arbitrary initial configurations to perform computations, has been investigated, with populations able to construct spanning trees in regular graphs using $O(\log D)$ memory, where D is the diameter of the graph [64]. Finally, populations protocols can be made to tolerate $O(1)$ crash failures, given some preconditions on the inputs [65].

Actively dynamic models can be classified depending on where the algorithmic control over the decision making lies. In *centralised* models, this control lies with a single

central computer which implements a centralised algorithm. The central computer usually has full knowledge of the state of the system. It therefore takes advantage of this to reconfigure the state of the programmable matter system with the benefit of global knowledge and the ability to efficiently co-ordinate nodes. The result is an optimal solution, provided that the problem is computationally tractable. For examples, see [19, 66]. By contrast, the control in *decentralised* models lies with each individual node, which must compute its actions and act independently of other nodes, for example by following a "look-compute-move" (LCM) cycle. In this setting, the ability of the nodes to co-ordinate and communicate information is emphasised. Examples include the aforementioned amoebot model [12] and the nubot model [11]. Real-life applications of programmable matter rely on decentralised models, as they usually do not have access to a single central computer, especially in use cases where the properties of programmable matter are valuable. However, the investigation of centralised algorithms for programmable matter is also valuable as they give a bound on what problems are feasible for decentralised algorithms to solve, as well as giving insights on how decentralised solutions to problems can be developed. They can also act as a simpler model for testing physical or mechanical properties from an engineering perspective without the need to give nodes computation and sensing capabilities. In this work, we explore the centralised setting with active dynamics.

In the future, as the cost of computational power continues to decline, there is the possibility that programmable materials capable of sensing, actuating, computing and communicating, will be mass-produced. This in turn could lead to applications as a form of highly sophisticated "smart material" [67]. Such materials would be able to change shape automatically and precisely, alternating between fluidity and rigidity as the situation requires. Items such as clothes and furniture would be able to change structure as the user desires. This will place emphasis on the ability of programmable matter models to represent the formation of complex 3D shapes. This could be accomplished with non-homogenous robots which specialise in forming separate components or are divided into categories based on properties such as material and size. Models may also have to incorporate more complex elements such as physical laws, as they move away from the abstract and towards the concrete representation of robots in physical space. From the perspective of passive systems, there is much recent work in dynamic networks using various models, and in the future there may be a focus on unifying these models into a single model which can represent every kind of network. There is also the possibility of using real systems as they are developed to inform the models of networks in highly dynamic environments [68].

In the following sections, we give a brief overview of our model and the results. The detailed definitions of the model, main properties and problems are presented in

Chapter 2, and the detailed analysis, related work and results are presented in the corresponding chapters.

1.2 Actively Dynamic Models

In this thesis, we consider an *actively dynamic* and *centralised* model where there is a set of n nodes which exist within *cells* on a square grid. Note that programmable matter models need not exist on a square grid, for example Walter *et al.* [69] uses a hexagonal grid, the amoebot model mentioned earlier uses a triangular grid, and there are even models using continuous 3D space [70]. These nodes are capable of making a single minimal movement, the rotation of one node around another. In addition, all nodes must form a single connected shape during and after every move. This *rotation* model was established in a previous work [31]. The motivation for this model is the possibility of representing systems of reconfigurable robots, with each robot only able to perform a minimal operation due to resource constraints, and dependent on connections with the other robots to perform tasks.

The problem we consider for the model in those chapters is the *RotC-Transformability* problem, established in [31]. In this problem, the nodes form an initial connected shape A on the square grid, and the goal is to transform the shape A into another shape B. This occurs via a series of rotation movements, with each rotation changing the *configuration*, the specific layout of the nodes on the grid. Time is represented sequentially as the number of rotation movements which the algorithm causes to occur. This is possible due to the assumption that the algorithm is centralised, with each node moving sequentially, and that computation time is negligible. If the nodes could make parallel moves, then a parallel representation of time would be necessary. More formally, RotC-Transformability is a centralised computational problem, where the goal is to decide for an arbitrary number of time steps t, the existence of a series of configurations $C_0, C_1, \ldots, C_t$, where $C_0 = A$, $C_t = B$ and for all C_i, for $0 \leq i \leq t$, the nodes form a connected shape. If this sequence exists, it returns the sequence of moves necessary to transition between the configurations. All rotation movements are *reversible*, a common property of actuation mechanisms in programmable matter models and a fact we take advantage of in our theorems. The rotation movement naturally divides the square grid into two sets of cells, resembling the checker pattern of a chess board, with nodes incapable of moving from one set to the other. We refer to these sets and the nodes which occupy them by the colours *red* and *black*.

The shapes we consider in Chapter 3 are those from a class called *nice shapes*, first defined in [71] as a class containing any shape S which has a central line L where for all nodes u in S either $u \in L$ or u is connected to L by a line of nodes perpendicular to L. Our goal is to solve the RotC-Transformability problem for any arbitrary pair of

nice shapes, so long as they are *colour consistent*, meaning the number of red and black nodes in both shapes is the same. This is a necessary condition for transformation, as nodes are incapable of changing their colour in the rotation-only setting. We first show that some shapes are *blocked*, meaning that they cannot meaningfully transform under the conditions of the problem. We therefore introduce a *seed*, also referred to as "musketeers", a group of nodes in a shape S which are placed in empty cells neighbouring a shape A to create a new connected shape which is the unification of S and A. Seeds allow shapes which are blocked or incapable of meaningful movement to perform otherwise impossible transformations. A seed consisting of x nodes will be called an x-node seed throughout the thesis. The use of seeds was established in a previous work [31], and more recently shown to enable universal reconfiguration in the context of connectivity preserving transformations [72]. By assuming a minimal seed of size 4, we show that there is a solution to RotC-Transformability for any pair of colour-consistent nice shapes.

In Chapter 4, we continue with the same model and problem, but this time we turn our attention to a class of shapes which we call *orthogonally convex* shapes, defined intuitively as all shapes where for each horizontal and vertical line containing nodes there is no empty cell between two cells occupied by nodes. This is a fundamentally different class of shapes to that of nice shapes, and more difficult to transform. It cannot easily be compared to the class of nice shapes. A diagonal line of nodes in the form of a staircase belongs to the class of orthogonally convex shapes but not the class of nice shapes. Any nice shape containing a gap between two of its columns is not an orthogonally convex shape. Finally, there are shapes like a square of nodes which belong to both classes. By moving to orthogonally convex shapes, we move towards universal transformation, because the class seems richer in structure than that of nice shapes thanks to the absence of a central line. If universal transformation turns out not to be possible, we at least move towards an improved understanding of which transformations are possible in the model.

Studies of systems of programmable agents on a grid which are capable of both rotation and sliding movements, where sliding is the movement of one node over two neighbouring nodes, have existed for over a decade. These are referred to as *metamorphic systems* [66, 73, 74]. An important property of these systems is the reversibility of movement, meaning that for any action a which causes the configuration C_0 to become C_1, there is a corresponding action b which causes C_1 to become C_0 again. In research on these systems, it is common for the problem of transforming shapes to be expressed as that of transforming from an initial shape A which belongs to a specific class of shapes, into a shape B from a designated class of shapes, which are referred to as *canonical shapes*. Since the movements are reversible, it trivially follows from any proof that an arbitrary shape in a class of shapes can transform into a canonical shape that

the reverse is possible, and therefore, that the transformation of any shape of the class under consideration into any other shape of the same class is possible, provided that they consist of the same number of agents. The most popular type of canonical shape is the line [69, 71, 75], which is also considered in Chapter 3 of our results, however there are other types of shapes such as squares [76], and we consider a *line-with-leaves* in Chapter 4. There are even recent efforts to avoid the use of canonical shapes for the sake of minimising shape-to-shape transformation complexity [77].

There are some important results in metamorphic systems which lead directly into our work. These systems are generally assumed to maintain connectivity. A first paper [73] focuses on moving shapes across the grid as fast as possible, and provides bounds on the speed of movement, defined using the number of cells traversed and time. It also does not focus exclusively on square grids, but considers hexagonal grids as well. A second paper [74] shows that the question of whether or not a motion maintains connectivity is decidable, and also whether or not a goal configuration is reachable from a given state. This in turn leads to the question of whether or not these systems are capable of *universal transformation*, the ability to transform any arbitrary shape into any other arbitrary shape with the same number of nodes. This was later shown to be possible in two independently developed proofs [31, 66]. One question left open by this result is whether or not universal transformation is possible with only one of the two movements considered.

The RotC-Transformability problem was defined as an extension of the *Rot-Transformability* problem, which is not constrained by the need to maintain connectivity. They introduce the concept of colouring nodes to represent the properties of rotation-only movement, which we used extensively in all our work in actively dynamic models. They also introduce the concept of blocked shapes, demonstrating that universal transformation is not possible with rotation only, leading to the necessity of the use of seeds. They introduce the Minimum-Seed-Determination problem, that of determining the minimum seed necessary to make the transformation between two shapes A and B feasible. They show that universal transformation is possible in the Rot-Transformability setting, provided that a 2-node seed is placed on the outside of the shape, and that it is in P. They then define the RotC-Transformability problem and show that RotC-Transformability is in $PSPACE$. They then move to a rotation and sliding model. As previously mentioned, they give a proof for universal transformation in this model, and then a $\Theta(n^2)$ time bound for transformation in the worst case. Finally, they show that it is possible to use parallelisation to speed up the transformation, leading to a $O(n)$ parallel time bound, where one time step in parallel time is equal to one movement for every node in the shape. We built on this work by developing transformation results in the RotC-transformability setting, with the ultimate goal being a universal transformation result similar to that in the Rot-transformability setting.

This left the open question of universal transformation with connectivity preservation, with preliminary work in an arxiv draft [78] of the paper. Our work continues from that open question towards the goal of demonstrating the feasibility of universal transformation in the setting. We believe this is a valuable goal for the development of programmable matter systems which rely on connectivity to be feasible, or for the completion of complex tasks.

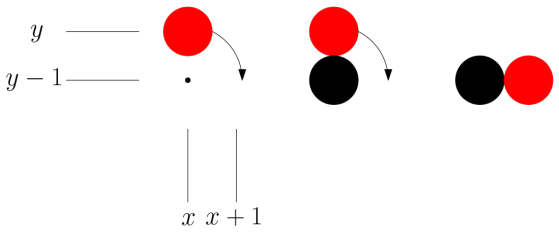


FIGURE 1.1: An example of a clockwise rotation movement. A node on the black dot (in row $y-1$) and empty cells at positions $(x+1, y)$ and $(x+1, y-1)$ are required for this movement.

There is also recent work into three dimensional systems, usually variants of systems which are more typically presented as occupying 2D space. This usually takes the form of extending results from the 2D model to a 3D variant, for example bounds on transformations imposed by symmetry have been extended to 3D models [79]. Results for the exploration of finite 2D square grids [80] have also been extended to 3D equivalents [81]. The parallel reconfiguration of robots in the crystalline robots model [82] has also been extended to a 3D model [19]. However, there are also problems unique to 3D models, for example the plane formation problem [70], where robots must all enter a common plane. It can been seen that results in 3D space are generally incidental extensions of 2D results, a necessary consequence of the focus on 2D systems which are generally easier to analyse. In the future, as more real-world applications for programmable matter systems are developed, there may be an increase in demand for solutions to problems encountered which cannot easily be solved by systems which assume existence within 2D space. This would naturally lead to a shift in emphasis towards 3D systems.

1.2.1 Centralised Transformations for Nice Shapes

As stated earlier, in Chapter 3, we consider the problem of transforming nice shapes into each other. We first show that there exist blocked shapes which cannot be meaningfully transformed. This is an extension of the blocked shapes set considered in [31], as it considers all shapes which are blocked due to the need to maintain connectivity, not just the line. We use this to justify the introduction of a seed to aid the transformation. We initially use a seed of size 3 to transform a line of nodes into a nice shape. We prove

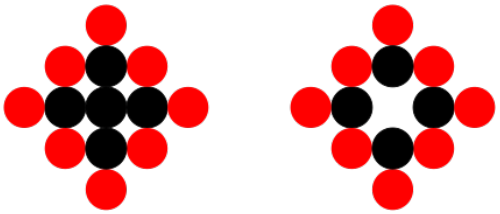


FIGURE 1.2: Two examples of blocked shapes.

this by using an algorithm, DepositNode, which uses the seed to raise nodes from the line and place them, constructing the nice shape node by node. We first prove this is possible for a subset of nice shapes where each of the lines perpendicular to the central line L are of even length. We then extend this result to all nice shapes, and show via lower and upper bounds that this takes $\Theta(n^2)$ time steps, and that this bound is optimal. However, because we must discard a node at the end of the transformation, it is not reversible. We therefore increase the size of the seed to 4 to get a reversible transformation of the line to any nice shape. It follows that by using the line as a canonical shape the transformation from any nice shape to any other nice shape is possible. The same time bounds apply to this transformation as well.

There were some technical challenges to overcome in the process of achieving this result. One of the first things to contend with when transforming nice shapes is that there are some which are *blocked*, meaning that they cannot meaningfully transform under the conditions of the problem. For example, a nice shape which consists solely of a line can only rotate the two ends of the line. It therefore naturally follows that some form of aid will be necessary if the transformation of any nice shape is to be accomplished in this setting. In addition, any strategy for the transformation of nice shapes must contend with the potential existence of gaps between the lines perpendicular to L. Furthermore, the central line L naturally bisects the shape in two, which hints at a strategy of treating each half separately. Our goal for this chapter was to show that there is a solution for these problems, one which we define algorithmically and describe with the help of figures.

1.2.2 Transformations for Orthogonally Convex Shapes

The goal in Chapter 4 is to show that the transformation of orthogonally convex shapes into each other is possible. We first define construction and destruction, which are respectively a series of node placements and removals, which either create a shape or completely remove it without breaking connectivity at any point in the process. We then show that a *6-robot*, a group of 6 nodes, is capable of traversing the perimeter of any orthogonally convex shape, and that the same holds for 7-robots. We do this by using a set of cases which represents all of the possible situations which the robots

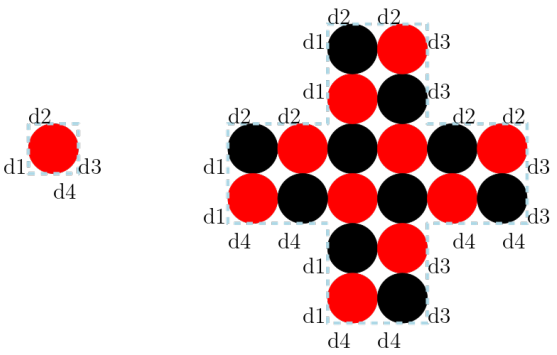


FIGURE 1.3: Two examples of orthogonally convex shapes. The blue line is the perimeter of the shape, used later in the paper.

can encounter when attempting to move, and demonstrating that it can always make progress, defined with the help of an invariant. We traverse the perimeter as the 6-robot, remove nodes from the shape according to a shape elimination sequence, traverse as a 7-robot, and place nodes according to a shape generation sequence. By repeating this process we can eliminate and generate shapes. We use this to transform an orthogonally convex shape into a shape we call the *diagonal line-with-leaves*, named after its resemblance to the line-with-leaves, a shape which consists of a line with extra nodes [31]. We use a diagonal version to maintain orthogonal convexity. We transform it via an intermediate shape which we call the extended staircase. We show this by using a series of algorithms. ExtendedStaircase creates the shape generation sequence for the extended staircase which will be built, OConvexToExtStaircase transforms the convex shape into an extended staircase, and ExtStaircaseToDLL transforms that into a diagonal line-with-leaves. By reversibility, we can transform orthogonally convex shapes into each other, treating the diagonal line-with-leaves as a canonical shape. There is a series of technical issues involved in showing that the transformation is feasible. For example, the connection of the extended staircase to the shape, which is not orthogonally convex but can still be traversed, as we prove in the chapter. The transformation begins with the addition of a minimal 3-node seed, to deal with blocked shapes as before. This seed forms a 6-node robot which performs the transformations by moving nodes from one cell to another, traversing the perimeter of the shape in the process. By following an algorithm which controls this process, it is possible to transform any orthogonally convex shape A into any other orthogonally convex shape B which is colour-consistent with A. We leave the extension of our methods to universal transformation, and the creation of a decentralised variant of our solution as natural open problems.

The results of this chapter are significantly more complex than those of the previous chapter, as a result of the need to traverse the perimeter of a shape which is not a trivial line, as it is in a nice shape which is under construction. In addition, there were a much larger number of edge cases than in the nice shapes scenario.

1.3 Roadmap

We consider centralised models in this thesis which allows us to make statements about feasibility. Chapter 3 and Chapter 4 contribute the transformation of classes of shapes into other classes of shapes within a programmable matter framework. The first concerns the transformation of nice shapes into other nice shapes, which is significant as it acts as a stepping stone towards universal transformation. The second shows that the transformation of orthogonally convex shapes into other shapes within the same class is possible, which is significant for moving much closer towards universal transformation, due to the absence of a central line allowing for much more complex structures. Both results reinforce each other in the sense that the two classes of shapes are not comparable.

In Chapter 2, we describe in detail the model of programmable entities that we used in our research. We also compare the model to other, similar models. In Chapter 3, we give the results of the first research question related to programmable matter, whether or not it is possible to transform a nice shape into another nice shape with the aid of a seed in the RotC-Transformability setting. In Chapter 4 we give the results of the second research question related to programmable matter, our investigation into the transformation of orthogonally convex shapes into other orthogonally convex shapes with the aid of a seed in the same RotC-Transformability setting. In Chapter 5 we conclude and give directions for potential future research.

1.4 Publications and Preprints by the Author Presented in this Thesis

Publications				
Chapter	Title	Authors	Conference	Journal
3	Centralised Connectivity-Preserving Transformations for Programmable Matter: A Minimal Seed Approach	Matthew Connor, Othon Michail, Igor Potapov	ALGOSENSORS (2021) [1][83]	Theoretical Computer Science (2021) [84]
4	Centralised Connectivity-Preserving Transformations by Rotation: 3 Musketeers for all Orthogonally Convex Shapes	Matthew Connor, Othon Michail	ALGOSENSORS (2022) [85]	Journal of Computer and System Sciences (2022) (invited, under review)

Table 1.1: List of author's publications and preprints presented in this thesis

[1]International Symposium on Algorithmics of Wireless Networks

Chapter 2

Model, Problems and Basic Properties

In this chapter, we provide all the notations and definitions which are used throughout this thesis. We begin by defining the centralised rotation model, used in Chapter 3 and Chapter 4. We then compare the rotation model to other, similar models.

We begin by formally defining in Section 2.1 the centralised model considered in this thesis, which we refer to as the rotation model, and provide definitions of the problems we later study. We provide definitions of the algorithmic concepts used by the model in 2.1.1, and follow this with some definitions of geometric concepts in 2.1.2. We then define blocked shapes, the set of shapes where meaningful transformation is not possible, in 2.1.3 as this justifies the use of seeds for transformations. We define the concept of canonical shapes, used in both chapters on the active model, in 2.1.4. We then define the target shapes of each chapter, nice shapes and orthogonally convex shapes, in 2.1.5. We define shape elimination and generation sequences in 2.1.6. Finally, in Section 2.2, we end with a comparison between the model and other, similar models in the literature.

2.1 The Rotation Model

2.1.1 Model and Problem Definitions

In the rotation model we consider in this thesis, programmable matter systems are organised via a graph $G = (V, E)$, where V is the set of nodes within a 2D square grid. Each cell is uniquely referred to by its $x \geq 0$ and $y \geq 0$ co-ordinates. The system itself consists of a set S of n modules, called *nodes* throughout. Each node may be viewed as a spherical module fitting inside a cell. At any given time, each node $u \in S$ occupies a cell in the grid $o(u) = (o_x(u), o_y(u)) = (x, y)$ (where x corresponds to a column and y to a row of the grid) and each cell can be occupied by at most one node at a time. At any

given time t, the positioning of nodes on the grid defines an undirected neighbouring relation $E(t) \subset S \times S$, where $\{u, v\} \in E$ iff $o_y(u) = o_y(v)$ and $|o_x(u) - o_x(v)| = 1$ or $o_x(u) = o_x(v)$ and $|o_y(u) - o_y(v)| = 1$, that is, if u and v are either horizontal or vertical neighbours on the grid, respectively. We say that two nodes are *edge-adjacent* if such a relation exists between them. A more informative and convenient way to define the system at any time t is the mapping $P_t : \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0} \to \{0, 1\}$, where $P_t(x, y) = 1$ iff cell (x, y) is occupied by a node. At any given time t, $P_t^{-1}(1)$ defines a shape. Such a shape is called *connected* if $(S, E(t))$ defines a connected graph. A *configuration* of a shape is an arrangement of the nodes of the shape on a 2D grid where each node is uniquely identifiable.

Let A and B be two connected shapes. We say that A transforms to B via a rotation r, denoted $A \xrightarrow{r} B$, if there is a node u in A such that if u applies r, then the shape resulting after the rotation is B. We say that A transforms in one step to B (or that B is reachable in one step from A), denoted $A \to B$, if $A \xrightarrow{r} B$ for some rotation r. We say that A transforms to B (or that B is reachable from A) and write $A \rightsquigarrow B$, if there is a sequence of shapes $A = C_0, C_1, ..., C_t = B$, such that $C_i \to C_{i+1}$ for all $0 \leq i < t$. Rotation is a reversible movement, a fact that we use in our results.

A line is a connected shape where every node lies on the same column or the same row. This basic shape is used frequently in our analyses.

Consider a black and red checkered colouring of the 2D grid, like that of a checkerboard. See Figure 2.1 for an example. Then any shape S consists of $b(S)$ nodes which lie on black cells and $r(S)$ nodes which lie on red cells. Two shapes A and B are *colour consistent* if $b(A) = b(B)$ and $r(A) = r(B)$. Because rotations are the only permissible move, it is impossible for a node to change colour. This is depicted in Figure 2.2. It follows from this that it is impossible to transform a shape A into a shape B if they are not colour consistent. For any shape S of n nodes, the *parity* of S is the colour of the majority of nodes in S. If there is no strict majority, we pick any as the parity colour. We use *non-parity* to refer to the colour which is not the parity.

We consider the problem of transforming shapes into each other via rotation movements. In this setting, this means there exists an initial shape A, and a centralised algorithm selects a node from A to be rotated as well as the rotation to be performed, generating the next configuration. Each individual rotation movement takes a time step, implying that time is sequential and that the number of movements and number of time steps to perform a transformation are equivalent. The number of time steps to transform shapes in the rotation only setting has a lower bound of $\Omega(n^2)$, following from Theorem 7 of [31] which provides the same bound for a stronger model. The centralised algorithm then terminates when the final configuration is reached, which is

equivalent to the target shape B. Transformation problems are therefore decision problems over whether or not a centralised algorithm exists which is capable of performing these transformations within the constraints of the problem.

We consider the following problems, first defined in [31]:

Definition 2.1. Rot-Transformability. Given a connected initial shape A and a connected target shape B, decide whether A can be transformed to B (equivalently, a translated and/or rotated variant of B) using only a sequence of rotation movements.

Definition 2.2. RotC-Transformability. The special case of Rot-Transformability in which A and B are again connected shapes and connectivity must be preserved throughout the transformation.

The ultimate goal in the long run is to prove universal transformation for these problems, meaning that both A and B are any arbitrary colour-consistent shapes. As already stated in Chapter 1, universal transformation for the first problem was already proven in the paper which defined it. Our main focus, therefore, is on the latter problem.

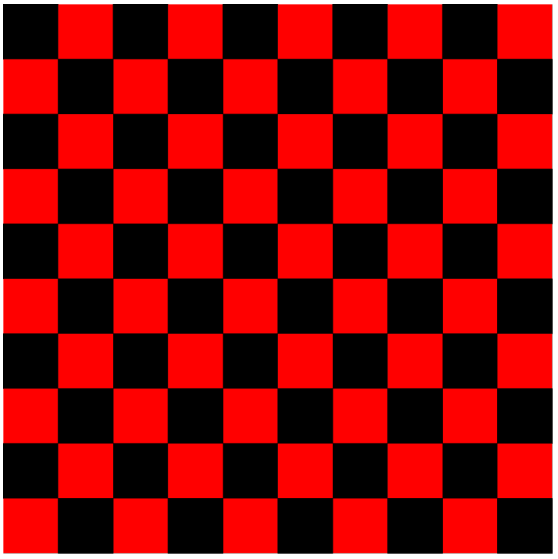


FIGURE 2.1: An example of the checkerboard pattern, used to represent the colouring of the 2D grid.

2.1.2 General Geometric Definitions

We now define some basic geometric notions which we will refer to extensively throughout this thesis. For example, the proofs in Chapter 4 require shapes to traverse other shapes, and in this section we define the geometry necessary to discuss this kind of movement.

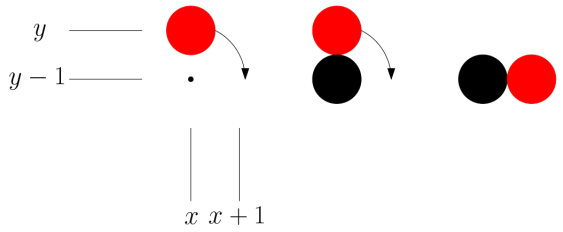


FIGURE 2.2: An example of a clockwise rotation movement. A node on the black dot (in row $y-1$) and empty cells at positions $(x+1, y)$ and $(x+1, y-1)$ are required for this movement. Red nodes, used throughout the paper, appear grey in print.

Definition 2.3. Let A be a connected shape. Mark each cell of the grid that is occupied by a node of A. A cell (x, y) is part of a hole of A if every infinite length single path starting from (x, y) (moving only horizontally and vertically) necessarily goes through an occupied cell. Mark every cell that is part of a hole of A as well, to obtain a compact shape of marked cells A'. Consider now polygons defined by unit-length line segments of the grid. Define the *perimeter* of A as the minimum-area such polygon that completely encloses A' in its interior. The fact that the polygon must have an interior and an exterior follows directly from the Jordan curve theorem [86].

All shapes considered in our results are compact, that is, lacking any holes, unless otherwise stated.

Definition 2.4. Any cell of the grid that has contributed at least one of its line segments to the perimeter and has not been marked (i.e., is not occupied by a node of A) is the *cell perimeter* of shape A. See Figure 2.3 for an example.

Definition 2.5. The *external surface* of a connected shape A, is a shape B, not necessarily connected, consisting of all nodes $u \in A$ such that u occupies a cell defining at least one of the line segments of A's perimeter.

Definition 2.6. The *extended external surface* of a connected shape A, is defined by adding to A's external surface all nodes of A whose cell shares a corner with A's perimeter.

2.1.3 Blocked Shapes and Infeasible Transformations

In this section, we cover a series of transformations which are infeasible, meaning that they rely on the ability to move $O(n)$ nodes but exist in a scenario where moving at most $O(1)$ is possible. We first define the class of shapes which are *blocked*, meaning there is no potential movement available for any node. We then define the class of *k-blocked shapes*, where the set of potential transformations has at most k configurations before any configuration is repeated. We only consider shapes for $k = 0$. Note however,

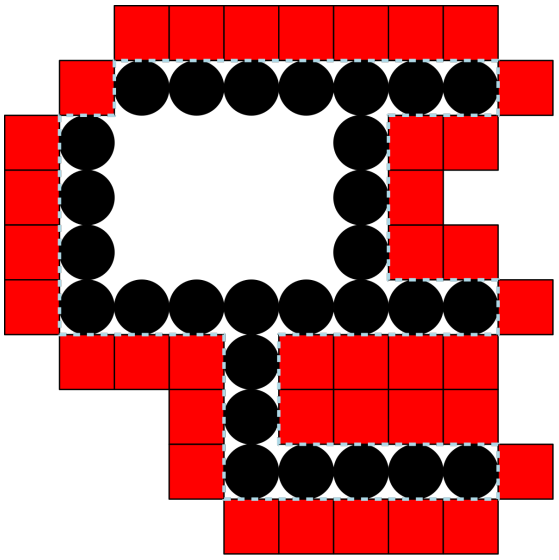


FIGURE 2.3: The light-blue dashed line is the perimeter of the shape. The red squares are the cell perimeter, or the set of empty cells which contribute at least one side to the perimeter. All nodes which share a side with the cell perimeter are part of the exterior, and all cells enclosed by the exterior constitute the interior.

that the end points of a straight line are blocked for $k = 2$, and the whole line for $k = 8$. We show that it is necessary for a seed to have at least 3 nodes if it is to be connected and to enable the movement of more than 5 nodes in a horizontal line. Finally, we provide a lower bound of $\Omega(n^2)$ movements for the problem of transforming a line into a nice shape.

Two nodes are *corner-adjacent* if their cells have corners which are adjacent to each other. A node w is an *interior* node if for each of the cells x edge-adjacent to w either there is a node occupying x or there are two nodes y and z such that y and z are edge-adjacent to x and corner-adjacent to w. A node is an *exterior* node if it is not an interior node. These relations are depicted in Figure 2.4. Note that when rotating nodes around each other, we use a special abbreviation, depicted in Figure 2.5.

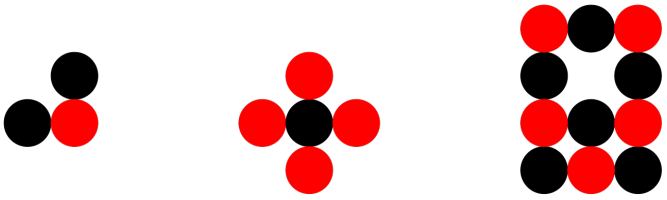


FIGURE 2.4: Examples of adjacency and interior/exterior nodes. The black nodes in the first image are edge-adjacent to the red node and corner-adjacent to each other. The black node in the second image is an interior node surrounded by exterior nodes. The black node in the middle in the third image is also an interior node.

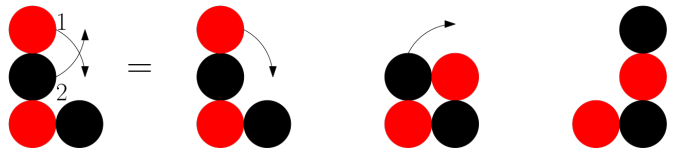


FIGURE 2.5: The rotation on the left is an abbreviated version of the rotations on the right, used throughout the paper. The numbers represent the order of rotations.

Lemma 2.7. *In the Rot-Transformability setting, a shape is blocked if and only if there is only 1 node or every exterior node has no edge connections to any other exterior node.*

Proof. A shape with one node is trivially blocked because there is nothing for it to rotate around. Otherwise, a shape consists of interior nodes connected to each other with the possibility of one-node gaps, surrounded by exterior nodes which form diagonal lines due to the edge-adjacency restriction. Interior nodes are blocked by the nodes that surround them, either because the grid space is filled by an edge-adjacent node or the two corner-adjacent nodes block the rotation movement. Exterior nodes can only rotate around nodes which are edge-connected, which must be interior nodes. The nodes which surround an interior node, whether edge or corner connected, always block an exterior node from moving, regardless of whether they are interior or exterior nodes themselves. Conversely, if there is an exterior node which is edge-connected to an exterior node, the exterior node can rotate into the empty space which it provides. □

This creates a shape which is similar to one or more overlapping rhombuses, for example Figure 2.6. Furthermore, with the additional condition of connectivity preservation, it is possible for these shapes to be connected by straight lines resembling a geometric cactus form of a cactus graph with these shapes instead of cycles.

Let S be an arbitrary shape with $B_1 \cup B_2 \cup \ldots \cup B_k = B \subseteq S$ as the set of all shapes which are blocked under the conditions of Lemma 2.7 which exist within S. Each shape B_i is *maximal*, meaning that $B_i \cup S'$ is a non-blocked shape, for all $S' \subset S$ edge-adjacent to B_i. Let $G(S)$ be a graph formed by first introducing one vertex for every $B_i \subset B$ and then one vertex for every other node in S. For all vertices u and v in $G(S)$, add an edge between them iff their corresponding nodes or blocked shapes in S are edge-adjacent.

Theorem 2.8. *An arbitrary shape S is blocked under the condition of connectivity preservation if the graph $G(S)$ is a tree, and every leaf in G is a blocked shape.*

Proof. By Lemma 2.7, each of the blocked shapes is incapable of movement. Because G is a tree, any vertex v in G which does not correspond to a blocked shape cannot be part of a cycle. Because every leaf is a blocked shape, all such v must be interior vertices with at least two edges. When a node rotates, it can only maintain edge connectivity with the

node it rotates around. Therefore, the rotation of any of the corresponding nodes would violate connectivity, equivalent to bisecting G into two disconnected components. □

We define a *connected* seed to be a seed which is a connected shape by itself. We next show that a connected seed of size $s < 3$ on a line of length n occupying the grid spaces $(0, 0)$ to $(n - 1, 0)$ can only move a constant number of nodes, 5 nodes to be precise. Note that if the seed is disconnected a 2-node seed is able to enable non-trivial movement by taking positions such that they can work with both ends of the line at the same time. The position of the seed can also be symmetrical so long as the destination of the pairs is also mirrored.

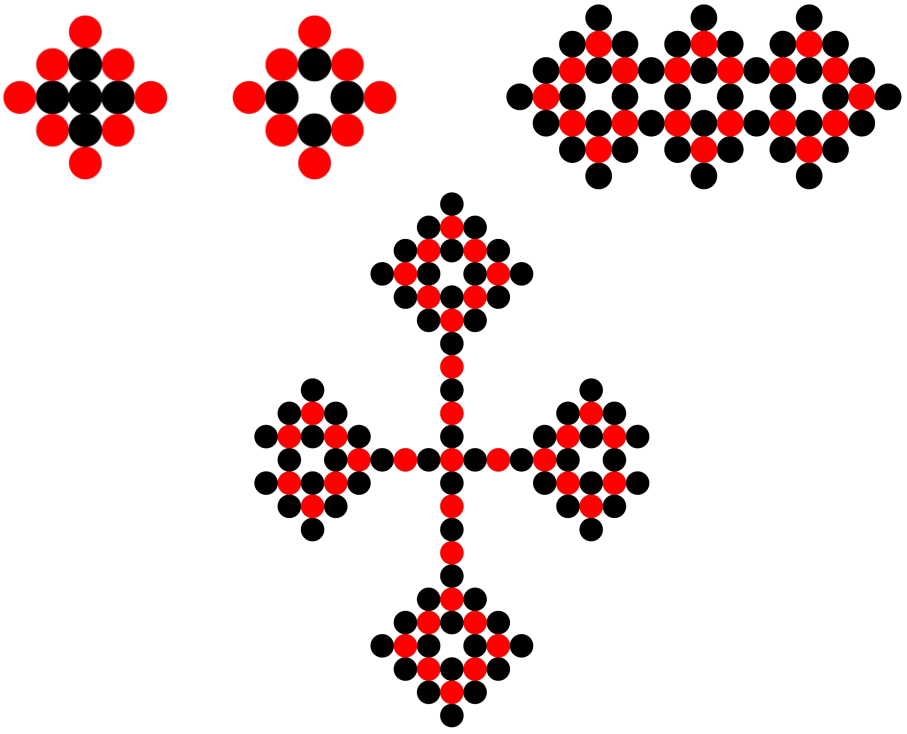


FIGURE 2.6: Examples of blocked shapes.

Lemma 2.9. *Any line of nodes S of length n can move at most five nodes from the line with any k-node seed of size $k < 3$ nodes.*

Proof. A line without seeds, with the connectivity preserving condition and with only rotation movements cannot do anything other than rotate the two nodes at each end point. With a 1-node seed, the only possible action is for the node to be positioned in the cell $(2, 1)$ (or any equivalent symmetrical position) and rotate the end node at $(0, 0)$ to $(1, 1)$ to form a pair. This is equivalent to having a 2-node seed on a line of length $n-1$. With a 2-node seed, it can only interact with an end node and with each node in the positions $(0, 1), (1, 1)$ or $(1, 1), (2, 1)$ (or any symmetrical position). In the former

case, the end node can only rotate around the node in $(0, 1)$ because it depends on it to maintain connectivity. In the latter case, the end node can rotate to $(0, 1)$. This allows the node in $(1, 0)$ and the node next to it (i.e. in $(2, 0)$) to rotate. However, they cannot move much without breaking connectivity thanks to a reliance on the nodes in $(3, 0)$ and $(3, 1)$ for connectivity which restricts movement.

Therefore, if we start with a 1-node seed, form a two node pair, rotate the node in $(0, 1)$ to $(2, 1)$, move the two nodes in $(1, 0)$ and $(2, 0)$ and the node at the other end of the line, we have exhausted all possibilities to maximise the number of moving nodes without using a seed of size $k \geq 3$. □

2.1.4 Canonical Shapes

When performing transformations between two arbitrary shapes in the same class, it is difficult to demonstrate that direct transformation is possible, as any algorithm which performs this must take any shape in the class as an input and return any other shape as output. *Canonical shapes* solve this problem by exploiting the property that any transformation is reversible. These shapes act as a single "target" shape for the shapes to transform into. If it is proven that any shape in the class can transform into the canonical shape, then it follows by reversibility that this shape can transform into any shape in the class, and since transformation is a transitive process it follows that transformation from any shape in the class to any other shape is possible. One key downside of the use of canonical shapes is that the transformation between shapes via such a shape may be much slower that the fastest transformation possible. Another is that a direct transformation between shapes may not need assistance in the form of a seed, but the transformation into the canonical shape does. In an extreme case, a transformation which requires only one rotation in the direct case may require a large seed and transformations between multiple shapes if transforming via a canonical shape.

An important property in the case of coloured shapes is whether or not the canonical shape is capable of being colour consistent with the target shape. For example, a line of nodes consists of alternating black and red nodes, with at most one more node of one colour than the other. This makes it incapable of acting as a canonical shape for shapes which are more imbalanced than that. In [31], the authors show that a connected shape with k black nodes has at least $\lceil (k-1)/3 \rceil$ and at most $3k+1$ red nodes. The same bounds hold when the colours are inverted. The maximum bound is the result of giving each black node 4 red nodes as neighbours, and having them share the minimal number of red nodes necessary to form a connected shape, which is $k-1$ for k black nodes. The lower bound is reached by inverting the colours in the same shape. This shape, which maximises the imbalance between the two colours, is called the line-with-leaves.

The two canonical shapes which we use in our work are the line, a series of connected nodes sharing the same column or row, and the diagonal line-with-leaves. The former can represent any shape where the number of black nodes and red nodes is equal or almost equal. We now show that the latter, a variant of the line-with-leaves, can represent any orthogonally convex shape.

Lemma 2.10. *For all $n \geq 3$, the maximum colour-difference of a connected orthogonally convex shape of size n is $n - 2\lfloor n/3 \rfloor$.*

Proof. We shall perform a column-based analysis of the maximum colour-difference of a shape S, assuming w.l.o.g. that the majority colour is black. By Proposition 2.16, every column is a consecutive sequence of nodes. This implies that every even-length column has an equal number of blacks and reds, and thus does not contribute to the colour-difference of S. It also implies that every odd-length column can contribute at most 1 (-1) to the colour-difference and a contribution of 1 (-1, resp.) happens iff that column starts and ends with a black (red, resp.), including the case of single-node columns. As a consequence, for a shape to maximise its colour-difference it must be maximising the number of black-parity odd-length columns while minimising the number of red-parity odd-length columns.

Consider any internal (i.e., which is not the leftmost or the rightmost) black-parity column c_x of S of length 1. Due to connectivity of S, the single black node (x, y) forming c_x must have the red neighbours $(x - 1, y)$ and $(x + 1, y)$. Note now that c_{x-1} and c_{x+1} cannot both have nodes above y nor both below y, as any of these would violate horizontal convexity of S. If only one of these two columns has additional nodes, then the contribution to the colour-difference by these 3 columns is 1 by using 5 nodes. If both columns have additional nodes, then let w.l.o.g. c_{x-1} have nodes above y and c_{x+1} below y. Then, again, the best contribution to the colour-difference is 1 by using 5 nodes, obtained by adding one black to each column. Adding more nodes to any of these cases cannot improve the 1/5 ratio.

Next, over all columns of odd length at least 3, the maximum contribution is obtained by the length-3 column $(black, red, black)$, which contributes to the colour-difference 1 per 3 nodes.

Consequently, given $n \geq 3$ nodes, a shape maximising the colour-difference is the one consisting of $\lfloor n/3 \rfloor$ columns of length 3 and $n - 3\lfloor n/3 \rfloor \leq 2$ terminal single-node columns, for a maximum colour-difference of $\lfloor n/3 \rfloor + n - 3\lfloor n/3 \rfloor = n - 2\lfloor n/3 \rfloor$, as required. This shape, which we call the *diagonal line-with-leaves*, is depicted in Figure 2.8. □

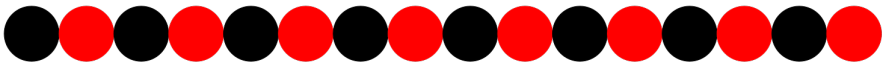


FIGURE 2.7: The line.

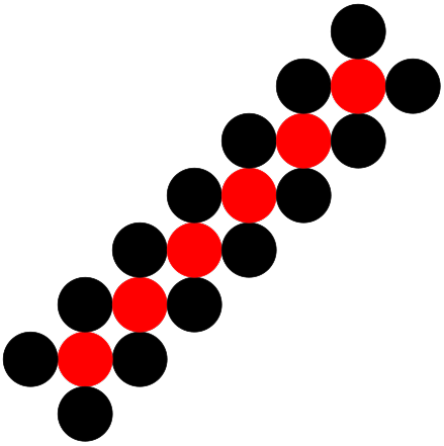


FIGURE 2.8: The diagonal line-with-leaves shape on n nodes, consisting of k red nodes and $2k$ black nodes arranged in $\lfloor n/3 \rfloor$ columns of length 3, plus at most 2 terminal single-node columns at the left and right ends.

2.1.5 Target Shapes

We now present the classes of shapes which are our target shapes, meaning that they are the shapes we wish to transform to and from within our setting. In Chapter 3, our goal is to construct nice shapes. These shapes were first defined by Almethen *et al.* in [71]. For an example, see 2.9. We define the concept of *waste*, which we use in Chapter 3 to allow for the transformation from one shape into another shape which is smaller than the first. This simplifies some of the transformations by removing the need to place every node from the initial shape in the new shape. We present the definition for this type of shape below.

Definition 2.11. If S is not a target shape and $S = A \cup B$ where A is a target shape, we call B the *waste* of the shape S and say that B is $|B|$ waste.

Definition 2.12. A nice shape N is defined as a shape which has a central line L where for all nodes u either $u \in L$ or u is connected to L by a line of nodes perpendicular to L.

Definition 2.13. Let $N_{n-w} = S \cup T$ be the class of shapes with n nodes, where S is a target shape of size $n - w$ and T is w waste.

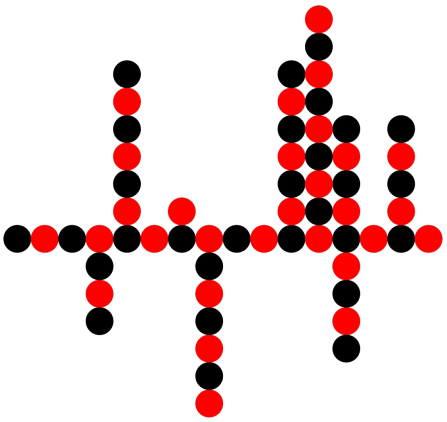


FIGURE 2.9: An example of a nice shape.

We now define *Orthogonally convex shapes*, the class of shapes considered in Chapter 4 together with some basic properties about them that will be useful later.

Definition 2.14. A shape S is said to belong to the family of *orthogonally convex* shapes, if, for any pair of distinct nodes $(x_1, y_1), (x_2, y_2) \in S$, $x_1 = x_2$ implies $(x_1, y) \in S$ for all $\min\{y_1, y_2\} < y < \max\{y_1, y_2\}$ while $y_1 = y_2$ implies $(x, y_1) \in S$ for all $\min\{x_1, x_2\} < x < \max\{x_1, x_2\}$.

Observation 1. Any discrete convex shape S is also orthogonally convex.

Observe now that the perimeter of any connected shape is a cycle drawn on the grid, i.e., a path where its end meets its beginning. This follows from the definition of a perimeter. The cycle is drawn by using consecutive grid-edges of unit length, each being characterized by a direction from $\{up, right, down, left\}$. For each pair of opposite directions, $(up, down)$ and $(left, right)$, the perimeter always uses an equal number of edges of each of the two directions in the pair and uses every direction at least once. For the purposes of the following proposition, let us denote $\{up, right, down, left\}$ by $\{d_1, d_2, d_3, d_4\}$, respectively. The perimeter of a shape can then be defined as a sequence of moves drawn from $\{d_1, d_2, d_3, d_4\}$, w.l.o.g. always starting with a d_1. Let also N_i denote the number of times d_i appears in a given perimeter.

Proposition 2.15. *A shape S is a connected orthogonally convex shape if and only if its perimeter satisfies both the following properties:*

- *It is described by the regular expression*

$$d_1(d_1 \mid d_2)^* d_2(d_2 \mid d_3)^* d_3(d_3 \mid d_4)^* d_4(d_4 \mid d_1)^*$$

  under the additional constraint that $N_1 = N_3$ and $N_2 = N_4$.

- *Its interior has no empty cell.*

Proof. We begin by considering the forward direction, starting from a connected orthogonally convex shape S. For the first property, the N_i equalities hold for the perimeter of any shape, thus, also for the perimeter of S. In the regular expression, the only property that is different from the regular expressions of more general perimeters is that, for all $i \in \{1, 2, 3, 4\}$, d_{i-2}, where the index is modulo 4, does not appear between the first and the last appearance of d_i.

Assume that it does, for some i.

Then d_{i-2} must have appeared immediately after a d_{i-1} or a d_{i+1}, because a d_{i-2} can never immediately follow a d_i. If it is after a d_{i-1}, then this forms the expression $d_i(d_{i-1} \mid d_i)^* d_{i-1} d_{i-2}$, which always has $d_i d_{i-1}^+ d_{i-2}$ as a sub-expression. But for any sub-path of the perimeter defined by the latter expression, the nodes attached to its first and last edges would then contradict Definition 2.14, as the horizontal or vertical line joining them goes through at least one unoccupied cell, i.e., one of the cells external to the d_{i-1}^+ part of the sub-path. The d_{i+1} case follows by observing that, in this case, the sub-expression satisfied by the perimeter would be $d_{i-2} d_{i+1}^+ d_i$, which would again violate the orthogonal convexity of S.

The second property follows immediately by observing that if (x, y) is an empty cell within the perimeter's interior, then the horizontal line that goes through (x, y) must intersect the perimeter at two distinct points, one to the left of (x, y) and one to its right. Thus, these three points would contradict the conditions of Definition 2.14.

For the other direction, let S be a shape satisfying both properties. For the sake of contradiction, assume that S is not orthogonally convex, which means that there is a line, w.l.o.g horizontal and of the form $(x_l, y), (x_l + 1, y), \ldots, (x_r, y)$, where (x_l, y) and (x_r, y) are occupied by nodes of S while $(x_l + 1, y), \ldots, (x_r - 1, y)$ are not. Observe first that any gap in the interior would violate the second property, thus $(x_l + 1, y), \ldots, (x_r - 1, y)$ must be cells in the exterior of the perimeter of S and (x_l, y), (x_r, y) nodes on the perimeter. There are two possible ways to achieve this: $d_3 d_2^+ d_1$ and $d_1 d_4^+ d_3$. These combinations are impossible to create with the regular expression, thus contradicting that S satisfies the properties. Similarly for vertical gaps. It follows that any shape fulfilling the two properties must belong to the family of connected orthogonally convex shapes. □

Let c_x denote the column of a given shape S at the x co-ordinate, i.e., the set of all nodes of S at x. Let $y_{max}(x)$ ($y_{min}(x)$) be the largest (smallest) y value in the (x, y) co-ordinates of the cells which nodes of a column c_x occupy.

Proposition 2.16. *For any connected orthogonally convex shape S, all the following are true:*

- *Every column c_x of S consists of the consecutive nodes $(x, y_{min}(x)), (x, y_{min}(x) + 1), \ldots, (x, y_{max}(x))$.*

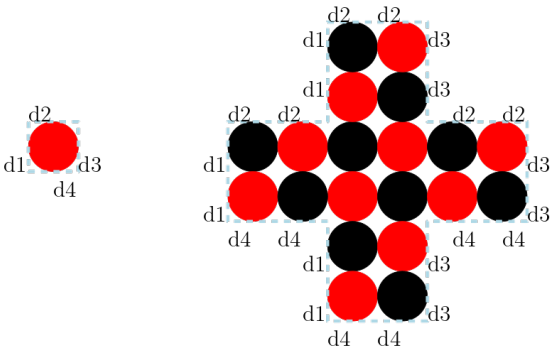


FIGURE 2.10: An example of two orthogonally convex shapes, with the directions of the perimeter labelled.

- *There are no three columns* c_{x_1}, c_{x_2}, *and* c_{x_3} *of* S, $x_1 < x_2 < x_3$, *for which both* $y_{max}(x_1) > y_{max}(x_2)$ *and* $y_{max}(x_3) > y_{max}(x_2)$ *hold.*
- *There are no three columns* $c_{x'_1}$, $c_{x'_2}$, *and* $c_{x'_3}$ *of* S, $x'_1 < x'_2 < x'_3$, *for which both* $y_{min}(x'_1) < y_{min}(x'_2)$ *and* $y_{min}(x'_3) < y_{min}(x'_2)$ *hold.*

All the above hold for rows too in an analogous way.

Proof. For the first property, observe that any discontinuity would violate vertical convexity of column c_x of S, thus, vertical convexity of S. Next, assume that the second property does not hold, that is, that there are columns c_{x_1}, c_{x_2}, and c_{x_3} of S, $x_1 < x_2 < x_3$, for which both $y_{max}(x_1) > y_{max}(x_2)$ and $y_{max}(x_3) > y_{max}(x_2)$ hold true. Let w.l.o.g. $y_{max}(x_3) \leq y_{max}(x_1)$. Then the horizontal line joining $(x_3, y_{max}(x_3))$ and $(x_1, y_{max}(x_3))$ passes through an empty cell above $(x_2, y_{max}(x_2))$, thus contradicting orthogonal convexity of S. A symmetrical argument holds for the third property. The proof for rows is identical, by rotating the whole system 90°. □

2.1.6 Elimination and Generation Sequences

Here we define some preliminary aspects of elimination and generation sequences for shapes. These are used extensively in our proofs for Chapter 4. They represent the addition and removal of nodes to shapes in the process of construction and deconstruction, and we exploit this to show that transformations which can simulate the generation and elimination sequences of shapes can construct and deconstruct those shapes.

For convenience, we define $F(S) = \{R_1, R_p, C_1, C_q\}$ as the set containing the first and last rows and columns of a given shape S (omitting S when clear from context), called *terminal* rows/columns, and *adjacent*: $F \to F'$, where $F' = \{R_2, R_{p-1}, C_2, C_{q-1}\}$, as a function mapping R_1 to R_2, R_p to R_{p-1}, C_1 to C_2 and C_q to C_{q-1}.

Recall that, by Proposition 2.16, every row/column of an orthogonally convex shape is a line.

Let S be a connected orthogonally convex shape. A *shape elimination sequence* $\sigma = (u_1, u_2, \ldots, u_n)$ of S is a permutation of the nodes of S satisfying the following properties. Let $S_t = S_{t-1} \setminus \{u_t\}$, where $1 \leq t \leq n$ and $S_0 = S$. Observe that S_n is always the empty shape. The first property is that, for all $1 \leq t \leq n-1$, S_t must be a connected orthogonally convex shape. Moreover, for all $1 \leq t \leq n$, u_t must be a node on the external surface of S_{t-1}. Essentially, σ defines a sequence $S = S_0[u_1]S_1[u_2]S_2[u_3]\ldots S_{n-1}[u_n]S_n = \emptyset$, where, for all $1 \leq t \leq n$, a connected orthogonally convex shape S_t is obtained by removing the node u_t from the external surface of the shape S_{t-1}.

A *row elimination sequence* σ of S is an elimination sequence of S which consists of p sub-sequences $\sigma = \sigma_1\sigma_2\ldots\sigma_p$, each sub-sequence σ_i, $1 \leq i \leq p$, satisfying the following properties. Sub-sequence σ_i consist of the $k = |R_i|$ nodes of row R_i, where $u_1, u_2, \ldots, u_k$ is the line formed by row R_i. Additionally, σ_i is of the form $\sigma_i = \sigma_i^1\sigma_i^2$, where (i) $\sigma_i^1 = (u_1, \ldots, u_k)$ or $\sigma_i^1 = (u_k, \ldots, u_1)$ and σ_i^2 is empty or (ii) there is a $u_j \in R_i$, for $2 \leq j < k$, such that $\sigma_i^1 = (u_1, \ldots, u_j)$ and $\sigma_i^2 = (u_k, \ldots, u_{j+1})$ or (iii) there is a $u_j \in R_i$, for $1 \leq j < k-1$, such that $\sigma_i^1 = (u_k, \ldots, u_{j+2})$ and $\sigma_i^2 = (u_1, \ldots, u_{j+1})$. We shall call any such sub-sequence σ_i an elimination sequence of row R_i. A *column elimination sequence* of S can be obtained by rotating the whole system by 90°.

Given a connected orthogonally convex shape S of n nodes, a *shape generation sequence* $\sigma = (u_1, u_2, \ldots, u_n)$ of S is a permutation of the nodes of S satisfying the following properties. Let $S_t = S_{t-1} \cup \{u_t\}$, where $1 \leq t \leq n$ and $S_0 = \emptyset$. Observe that $S_n = S$. Any shape generation sequence also satisfies the following properties, which it shares with the shape elimination sequence. The first property is that, for all $1 \leq t \leq n-1$, S_t must be a connected orthogonally convex shape. Moreover, for all $1 \leq t \leq n$, u_t must be placed in the cell perimeter of S_{t-1}. Essentially, σ defines a sequence $\emptyset = S_0[u_1]S_1[u_2]S_2[u_3]\ldots S_{n-1}[u_n]S_n = S$, where, for all $1 \leq t \leq n$, a connected orthogonally convex shape S_t is obtained by adding the node u_t to the cell perimeter of S_{t-1}.

2.2 Model Comparison

The most obvious model to compare to the rotation model is the rotation and sliding model. This model was established in pushing squares around [66], which showed that universal transformation is possible in $O(n^2)$ sequential time in the model, and a recent paper [31] has shown that universal transformation can be parallelised and solved in $O(n)$ parallel time. There are also results for heterogenous systems with the same movement, which have also achieved $O(n^2)$ sequential time transformations with a centralised model [87], by which they argue that space and not time is the "critical resource" for solving reconfiguration problems.

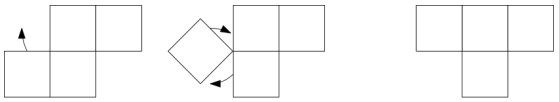


FIGURE 2.11: An example of a pivot movement.

A second type of model which is similar to the rotation model is a model with pivoting movements [88, 89], which represent nodes using squares which pivot on a single corner when moving cells. For an example, see Figure 2.11. Pivoting movements take more space to perform than rotations, as the pivoting node must temporarily enter cells close to the movement. However, pivoting allows nodes to replicate slide movements, and therefore nodes which pivot can move between cells of a different colour, unlike in the rotation model. A recent result [72] showed that the pivoting model also has blocked shapes, and when a shape is aided by up to 5 additional nodes and connectivity is preserved through the corners of nodes, universal transformation is possible in $O(n^2)$ time.

Chapter 3

Centralised Connectivity-Preserving Transformations for Nice Shapes

3.1 Introduction

In this chapter, we consider the *rotation* model, an *active* and *centralised* model where there is a set of n nodes which exist within *cells* on a square grid and are only capable of rotating around each other. Each time step, a single node rotates once.

The goal is to transform an initial shape A into the target shape B without breaking connectivity. The specific shapes we consider in this chapter are nice shapes, defined, as in Chapter 2, as a shape S which has a central line L where for all nodes u in S either $u \in L$ or u is connected to L by a line of nodes perpendicular to L. Our goal is to solve the RotC-Transformability problem for transformations between any arbitrary pair of nice shapes, so long as they are colour consistent, previously defined as the condition that the number of red and black nodes in both shapes is the same.

One of the first things to contend with when transforming nice shapes is that there are some which are blocked and cannot meaningfully transform under the conditions of the problem. For example, no nodes in a rhombus shape can perform any rotation movements. Also, some shapes are blocked in bounded sets of reachable configurations due to the need to maintain connectivity, for example, a nice shape which consists solely of a line can only rotate the two ends of the line. It therefore naturally follows that some form of aid will be necessary if the transformation of any nice shape is to be accomplished in this setting.

We assume the existence of a seed, to allow shapes which are blocked or incapable of meaningful movement to perform otherwise impossible transformations. The use of seeds was established in a previous work [31], and more recently shown to enable

universal reconfiguration in the context of connectivity preserving transformations [72], however to our knowledge there has been no attempt to investigate this problem using a seed which is a connected shape fully introduced before the transformation is initiated.

Any strategy for the transformation of nice shapes must contend with the potential existence of gaps between the lines perpendicular to L. In addition, the central line L naturally bisects the shape in two, which hints at a strategy of treating each half separately. Our goal for this chapter is to show that there is a solution for these issues, one which we define algorithmically and describe with the help of figures.

We show that it is possible to transform such a line into a nice shape of $n-1$ nodes using a 3-node seed in $O(n^2)$ time. We then demonstrate that it is possible to transform nice shapes of size n into other nice shapes of size n by using the canonical shape of a line and a 4-node seed in $O(n^2)$ time. We provide an algorithm to implement this transformation and give time bounds for it.

In the next section we give our algorithm for the construction of nice shapes where the colour of nodes added to each side of the line always alternates, then generalise to all nice shapes.

3.2 Transformation for Nice Shapes

In this section, we investigate the possibilities related to the transformation of shapes which are connectivity preserving. We focus on the problem of converting a nice shape of $O(n)$ nodes into any other nice shape of $O(n)$ nodes using an $O(1)$ seed. We do this by showing we can transform the canonical shape of a line with $O(n)$ nodes into any nice shape. Due to reversibility, it follows that any nice shape can be transformed into such a line, and then into another nice shape. More specifically, we first provide a solution for the variant of this problem (which we call M) where all the lines perpendicular to a central line L in the nice shape are such that the node at the end of each line is the opposite colour to the node at the end of its nearest neighbouring lines. We then prove that slight modifications to the method of construction allow for the class of all nice shapes to be constructed. Our methods construct a shape which is a union of a nice shape with constant waste $O(1)$.

We start with a shape S which is a line of length n occupying the cells $(0,0)$ to $(n-1,0)$. We are allowed to attach at an arbitrary position a k-node seed forming an arbitrary connected shape of size k to the line. We use a 3-node seed as this is the minimum size which allows us to move more than 5 nodes without breaking connectivity. This initial configuration is depicted in Figure 3.1. It is possible for our results to apply to a disconnected 2-node seed with a slightly modified procedure but with higher waste. We place the seed in a specific position as the connected 3-node seed is incapable of movement. We sketch the line to nice shape proof in the following subsection.

FIGURE 3.1: The line with the seed attached.

Definition 3.1. Let $N_{n-w} = S \cup T$ be the class of shapes with n nodes, where S is a nice shape of size $n - w$ and T is w waste.

Definition 3.2. Let $M_{n-w} = S \cup T$ be the class of shapes with n nodes, where S is a nice shape of size $n - w$ and T is w waste for which the following properties hold: For all lines $l_1, l_2, \ldots, l_k$ perpendicular to L, where L is the central line of S, the node at the end of each l_i, where $1 < i < k$, is the opposite colour to the node at the end of the lines l_{i-1} and l_{i+1}. For l_1 and l_k, The node at the end should be the opposite colour to the node at the end of l_2 and l_{k-1}, respectively.

3.2.1 Line to Nice Shape

We begin by considering the transformation of a line into a shape from M_{n-w}. The restriction of this class guarantees that no node with the "wrong" colour is ever in the position to block construction. The process of construction is therefore simpler. A method of dealing with these nodes, introduced later, will allow the restriction to be dropped, yielding the construction of N_{n-w}. Our first result is the following theorem:

Theorem 3.2. *A line of length n can be transformed to any given nice shape in the class M_{n-1} using a 3-node seed in $O(n^2)$ time.*

To solve this problem, we follow a strategy of having nodes rotate onto the horizontal line with the help of the 3-node seed and then constructing lines perpendicular to the horizontal line using the nodes. Additionally, we move 4 nodes below the line and on the opposite side to the seed. These nodes can then replicate the behaviour of the seed on the other side of the line, allowing for construction to occur below as well as above the line. Because their behaviour is the same, we refer to the seed and the group of nodes on the other side of the line as *builders*. As a result, the horizontal line becomes the central line L of the nice shape, and the vertical lines become the lines of nodes perpendicular to L. Finally, the seed and a single node which aid construction cannot be incorporated into the final shape and are discarded as waste.

To prove that this is possible, we define three algorithmic procedures. The first procedure, *RaiseNodes*, allows a builder to move two nodes at a time from the horizontal line. These nodes combine with the builder to form a 5-node cluster. This cluster can be broken if necessary into a 3-node line and a 2-node line, allowing the 2-node line to move by having each node rotate around the other. The second procedure, *MirrorSeed*, is

the procedure for creating the second builder below the horizontal line. It accomplishes this by moving two of the 2-node lines to the end of the horizontal and then rotating nodes in such a way that the four nodes are "pushed" through the horizontal and to the other side. The final procedure, *DepositNodes*, collects nodes from the horizontal line and deposits them in any reachable location. We will show that the set of reachable locations enables the construction of any nice shape.

As a result, we end up with nice shapes where the central column L corresponds to what is left of the original horizontal line. However, the resulting nice shapes $\mathcal{M} \subset \mathcal{N}$, where $\mathcal{N}$ is the set of all nice shapes, have only even lines. This is due to the construction procedures, which place two nodes at a time. We therefore provide additional movements that allow us to expand the set of nice shapes which can be constructed to include all nice shapes. We perform this for a special case and then generalise to drop this assumption and get any nice shape.

3.2.2 RaiseNodes

We use a 3-node seed in the cells $(2,1)$, $(3,1)$, $(4,1)$ for our operations as, by Lemma 2.9, a two node seed is incapable of helping nodes to move.

We call the first operation *RaiseNodes*. For this operation we use the 3-node seed to move nodes from the horizontal line such that they are on top of the horizontal line. In the process, the 3-node seed moves along the horizontal line such that each node moves from its original position $(x,1)$ to $(x+2,1)$.

We can raise two nodes at a time as a pair. The result can also be interpreted as a shape consisting of 5 nodes, which we refer to as a 5-node seed. Moving the pair of nodes once they are on the line is a trivial process. Each node rotates around the other node, alternating their relative positions within the two node shape.

The following lemma shows that these operations are possible.

Lemma 3.2. *Using a 3-node seed in the cells* $(2,1)$ *to* $(4,1)$*, it is possible to move 2 nodes from the line such that the 3-node seed is converted into a 5-node seed.*

Proof. The seed can only be placed in the cells specified as a three node line is incapable of translating to another position.

First, the leftmost node of the horizontal line at $(0,0)$ must rotate above the line. Then the third node in the seed at $(3,1)$ can rotate right, creating a space for the node just raised from the line, which then takes its place. By moving the two nodes in $(4,1)$ and $(2,1)$ one space right in the same manner, the four nodes above the line have moved two spaces to the right, creating room for another node to be raised from the line. □

Figure 3.2 depicts the process.

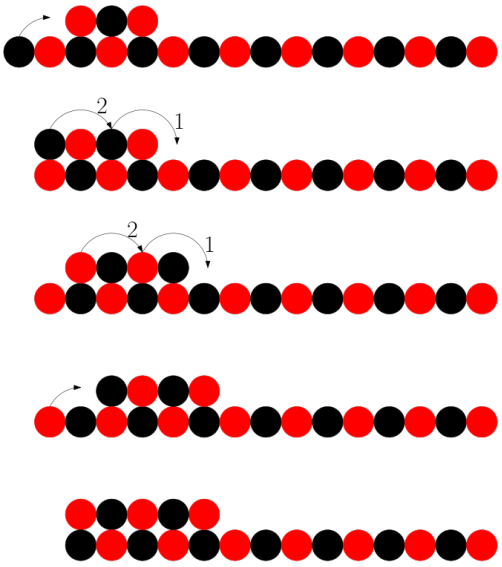


FIGURE 3.2: Raising nodes from the line. All numbers refer to the sequence of operations. Multiple rotations around the same node are represented by a single long arrow.

In addition, we can move the 5-node line to the right by following a series of specific rotations (see Figure 3.3). Furthermore, the technique of moving the 4-node line to the right from Lemma 3.2 can be used to move any line of even length. We can therefore move any line of odd length to the right by first splitting it into a 5-node line and a line of even length and then moving them separately. As a result, the process of raising nodes from the line can be repeated indefinitely so long as the nodes on the line have the space to be moved out of the way of the operation.

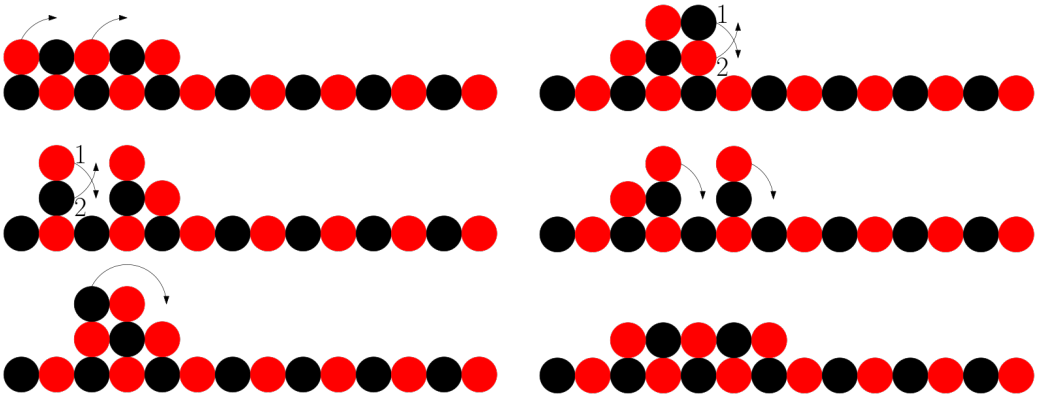


FIGURE 3.3: Moving a line of 5 nodes. Figures should be read vertically.

3.2.3 MirrorSeed

We now use RaiseNodes for our next operation, *MirrorSeed*, to place four nodes at the opposite side of the line (i.e. $(n-4,1)$ to $(n-1,1)$) and then push them through and below the line, creating a four node mirror of our original seed in the cells $(n-4,-1)$ to $(n-1,-1)$. Having a mirror of the original seed allows us to perform construction operations on the bottom of the horizontal line. We do this in 3 steps: raise four nodes using RaiseNodes twice to create a 7-node line, position four of the nodes at the end of the line and rotate the nodes and those at the end of the line such that the four nodes move through (not around) the line and to the other side.

Lemma 3.3. *Using a 3-node seed in the cells* $(1,1)$ *to* $(3,1)$*, above a line* L *of length* n *it is possible to create a 4-node line in the cells immediately below the nodes* $(n-4,0)$ *to* $(n-1,0)$*.*

Proof. We first move the 4 leftmost nodes in S, S_0 to S_3 to the top of the line. We do this by raising S_0 and S_1, and then repeat the procedure a second time with the next two nodes S_2 and S_3. We now have 4 nodes a square above the end of the line. By rotating them around each other in pairs we can place them in the cells $(n-4,1)$ to $(n-1,1)$. We can then "push" the nodes to the other side of S by following the procedure depicted in Figure 3.4. The result is four nodes in the cells $(n-4,-1)$ to $(n-1,-1)$ □

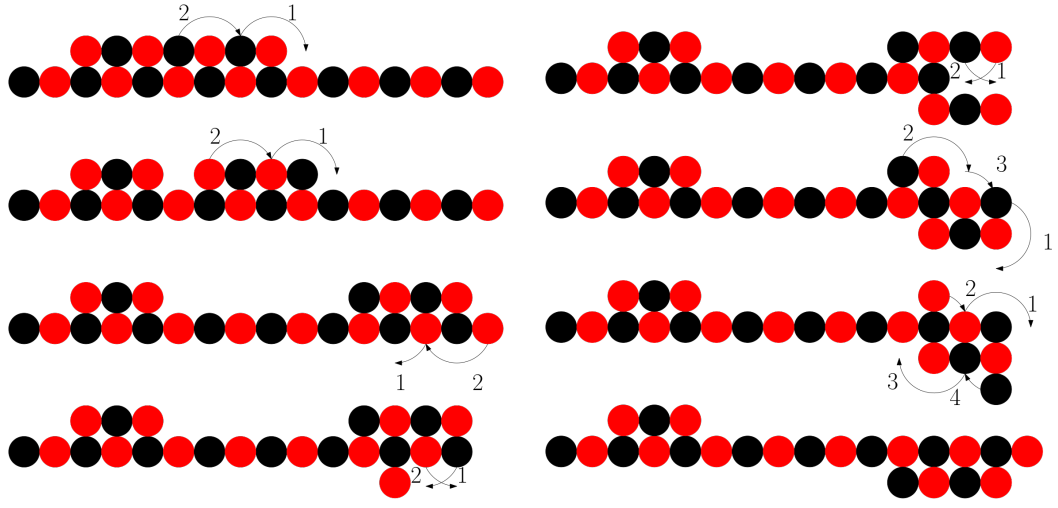


FIGURE 3.4: Pushing the nodes through the line.

3.2.4 DepositNode

Next, we present *DepositNode*, a sub-procedure using the 3-node seed to create a 5-node shape and move a node from the horizontal line to any empty cell which the shape can reach, provided the 5-node shape has the correct colouring, defined as having 3 nodes of the colour which will fill the cell.

We raise two nodes from the line, use this shape to deposit a node and move the other 4 nodes as a square back to the left. By leaving the cells above and below the two leftmost nodes in the line empty we can rotate the leftmost node, merging with the square to create a new 5-node shape. We can therefore repeat the process of moving for each node one at a time. In addition, this sub-procedure can be applied to the builder on the other side of the line.

We provide pseudo-code to describe this process. We input an instance of the nice shape which is under construction and the co-ordinates of a destination. The algorithm then moves the nodes such that a node is placed in the destination by the relevant builder, creating the output. In this way, the construction of the nice shape takes place over a series of phases, where each phase $1 \leq x \leq j$ corresponds to the movement of the xth node to the xth destination.

If the seed nodes are making their first transfer then they need to raise two nodes to become a builder with 5 nodes. After that, they only need to raise one node at a time. getLineHead() gets the node which is currently leftmost in the line. move() causes the builder to transfer one of its node to the destination. It must be the same colour but does not need to be the exact node. getParity() gets the colour of the node which must be placed first. rotateBlacks(seed) rotates each of the black coloured nodes in the seed rightwards. badParity occurs when the leftmost node of the line is not the colour which must be placed next. This situation represents a worst case scenario for node placement.

Our strategy is to demonstrate that the moves each builder can make are sufficient to be able to construct a nice shape. For ease of understanding, we provide visual representations of the movement we intend to accomplish. In this example, we show that it is possible to deposit the node at the end of the horizontal line.

Lemma 3.4. *A 3-node seed on any line S of length n, where n is an even number, can transfer a node the other end of the line.*

Proof. We position the 3-node seed in the same grid spaces as in Lemma 3.3. We can then follow the process in Figure 3.5 to achieve our result. Note that we must raise three nodes from the line to deposit a node successfully, as we must be able to choose which colour is deposited first. We do this by using the RaiseNodes process such that three nodes are raised from the line. The other node is then the first node selected by getLineHead(). The process of moving right two spaces is repeatable, these repetitions

Algorithm 1 DepositNode

```
1: destination ← (x, y) //co-ordinates node will be deposited on
2: //If this is the first transfer for the seed nodes
3: if (badParity) == true then
4:     pair ← RaiseNodes
5:     //If the first node must be red
6:     if getParity() == "red" then
7:         //the node at the front when the seed is formed
8:         node ← seed[3]
9:     else
10:        //rotate the black nodes to place one at the front
11:        rotateBlacks(seed)
12:        //the node at the front after the rotation
13:        node ← seed[2]
14: else
15:    //the current leftmost node of the line
16:    node ← getLineHead()
17: move(node, destination)
```

are omitted. This entire process can be performed symmetrically by the nodes at the bottom of the line by only raising two nodes from the line. □

In this way, we can place a node of the colour we prefer onto both sides of the line. It is then possible to (see Figure 3.6) transfer the builder to a vertical line. By positioning the builder carefully we can ensure that the movement is equivalent to crossing a line of even length. Therefore the process of adding another node can be performed on vertical lines, such as the ones we will build for our nice shapes.

To build any vertical line, we must first show that it is possible for DepositNode to construct lines of length 4 above the horizontal line. After that, because it is possible for the builders to shift onto a 4-node line, the situation becomes that of depositing a node at the end of a line.

Lemma 3.5. *Using a 3-node seed in the cells* $(1, 1)$ *to* $(3, 1)$*, above a line* L *of length* n *it is possible to create another line of length* 4 *above any* $u_i \in L$*.*

Proof. For this situation we have two scenarios: one where the colouring is correct and another where it is incorrect. We first consider the correct colouring and then show how to deal with the incorrect colouring.

For the first node, we simply deposit the node using DepositNode above u_i. The next node is deposited above u_{i-1} and rotated to be above the first. The next two nodes are more difficult, so we have provided Figure 3.7 to illustrate the process.

In the case where the colouring is incorrect, we deposit the incorrect node anywhere to the right directly above L and collect a second node from L. We can then merge the

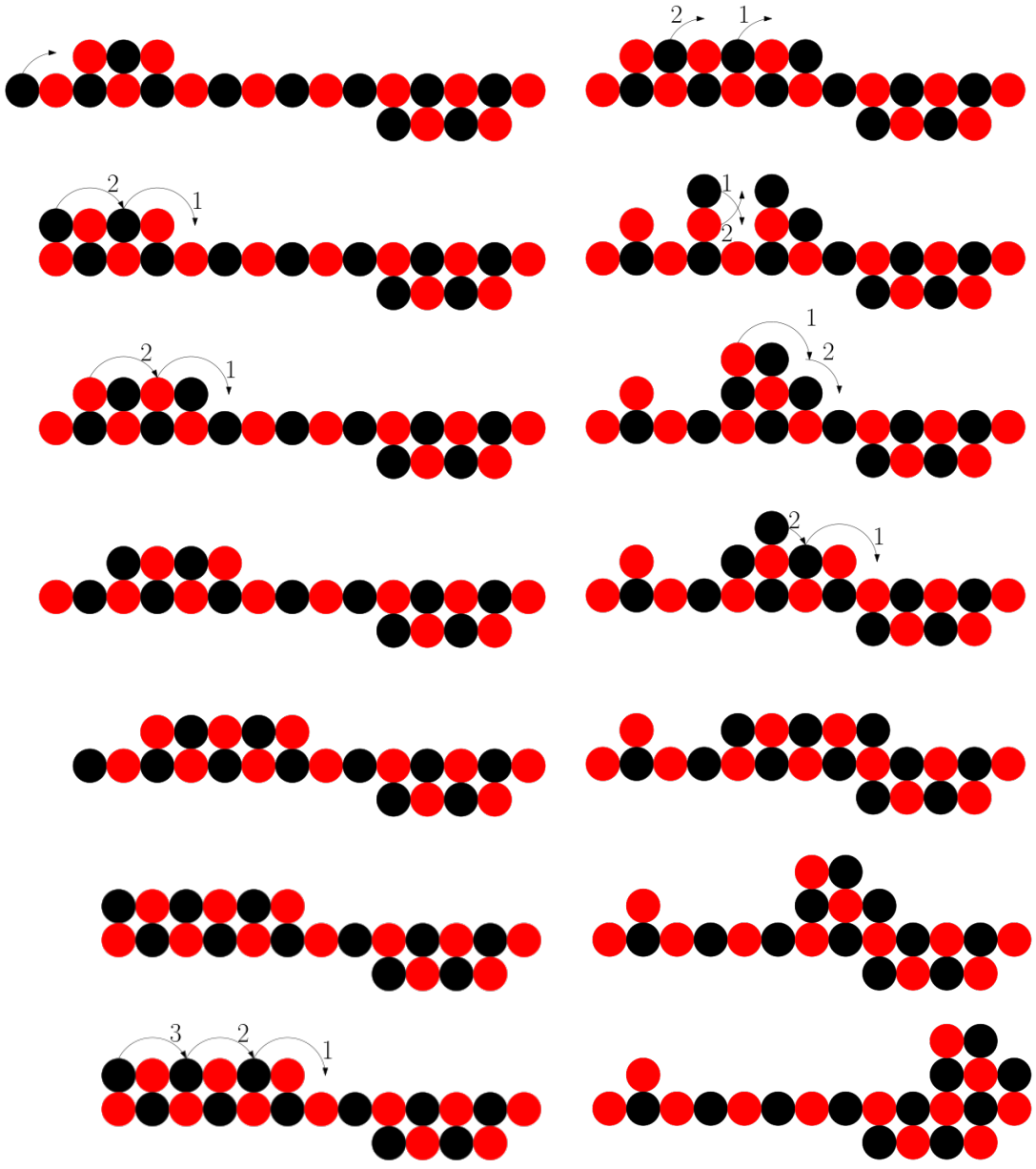


FIGURE 3.5: Moving a node across the line. To reach the second configuration in the left column, we raise three nodes by using the RaiseNodes process twice and stopping the second process early. In this figure, we place the black node. If we want to place the red node somewhere on the line, we can omit the first rotation in the second subfigure and create a 5-node seed with a red node at the front.

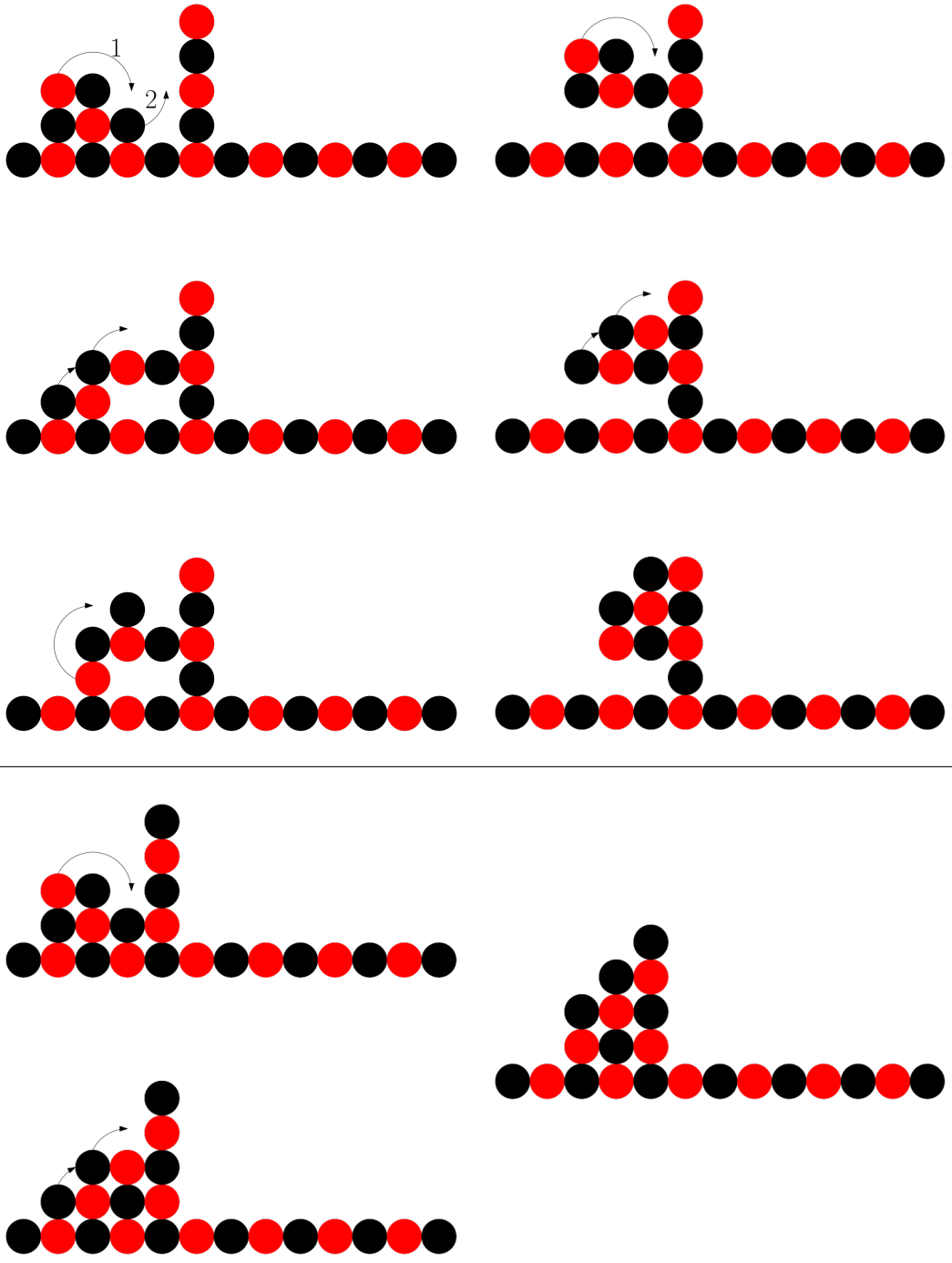


FIGURE 3.6: An example of moving the builder onto a vertical line. When the sequence of colours in the line is more convenient, a much simpler process can be used.

5-node shape with the node we deposited temporarily to create 6-node shape. Then 5 nodes of the correct colouring can split from the shape we created and deposit the node.

The 4-node square can then return to the node that was left behind and use it as the next node for depositing. In this way, the 5-node shape is capable of "selecting" its colouring. □

The next step we must take is to show that we can select the parity of the construction. This is a necessary process to guarantee that the initial processes of construction do not result in a scenario where two nodes of the same colour must be placed at the same time.

Lemma 3.6. *Using a 3-node seed in the cells* $(1, 1)$ *to* $(3, 1)$*, above a line* L *of length* n *it is possible to construct a two node line above* L *regardless of the colour of the node in* $(0, 0)$*.*

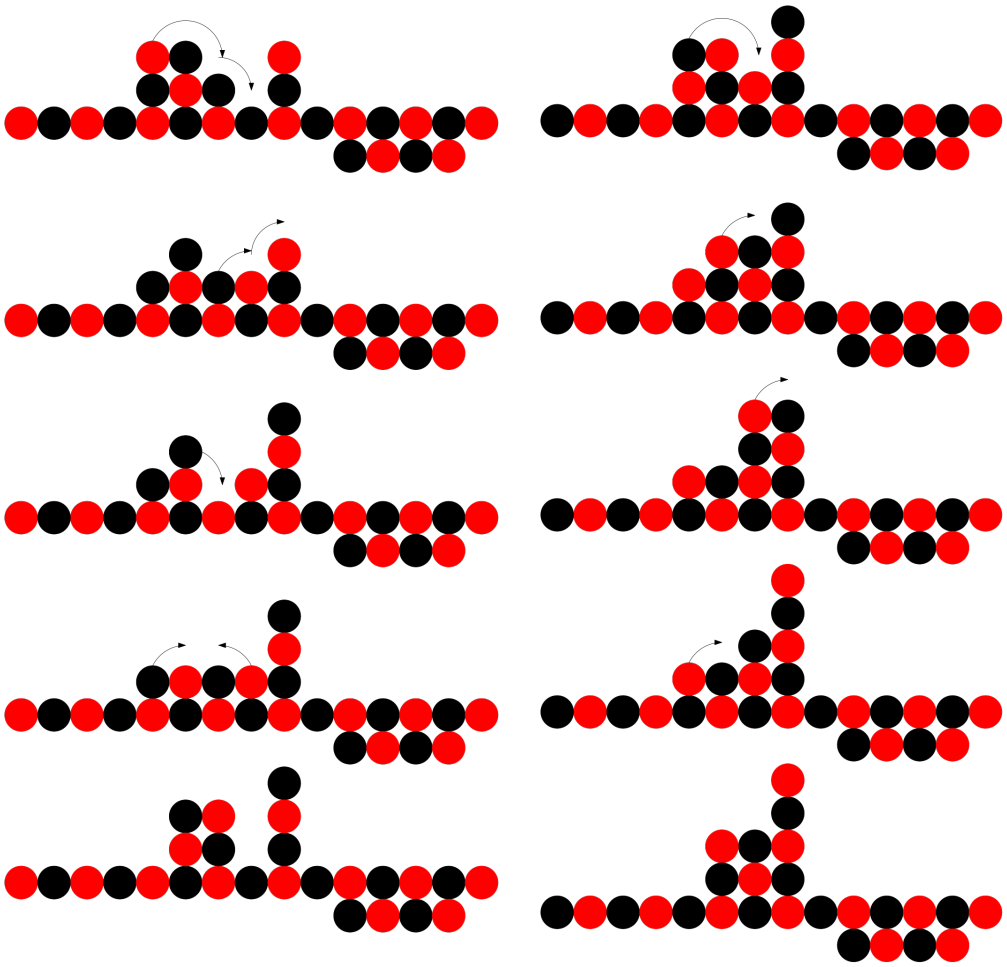
Proof. We begin by raising two nodes from L. We then use four of these nodes to form a 4-node square. We do this such that the node which is not part of the square is the opposite colour to the node at the end of L (i.e. in $(2, 0)$) We then have two scenarios depending on whether the first node we intend to place is the colour of the node on the line or the node in $(2, 0)$. If it is the same as the node on the line, we place the node on the line in the correct position and then place the node from the line above it. If the first node is different, we take the node from the line and form a 6-node block with the other five on the line. We can then place both nodes in the correct order. □

3.2.5 Construction of a Subset of Nice Shapes

We now have all of the lemmas that are necessary to prove that it is possible to construct a specific subset of the class of nice shapes. We first present an upper bound on the time for constructing nice shapes using our algorithm. We then prove that using our sub-procedures we can construct a nice shape using a line and a 3-node seed, and finally we show that process is reversible using a 4-node seed.

Lemma 3.7. *The transformation of a horizontal line of* n *nodes into any nice shape requires* $O(n^2)$ *time steps.*

Proof. The RaiseNodes and MirrorSeed algorithmic procedures perform a sequence of specific movement and as such are $O(1)$ time. The DepositNode procedure moves 5 nodes, deposits a node and returns with 4 nodes. Therefore we must perform $5n + 4n$ moves to transfer one node, and in the worst case, we must build a vertical line of length n above the last node $(n - 1, 0)$ in the horizontal line. This means each node must move past $n - 1$ nodes to reach their destination.



(A) Adding the third node

(B) Adding the fourth node

FIGURE 3.7: Adding the last two nodes.

Therefore the process is bound by the speed of DepositNode, which is $(5n + 4n) \cdot (n - 1) = O(n^2)$ time steps. □

Theorem 3.8. *A line of length n can be transformed to any given nice shape in the class M_{n-1} using a 3-node seed in $O(n^2)$ time.*

Proof. The seed is positioned above the second, third and fourth nodes in the horizontal line, at $(1, 1)$, $(2, 1)$, and $(3, 1)$. We first use RaiseNodes twice to raise 4 nodes from the line and then use MirrorSeed to create a 4 node builder below the horizontal line. Then, we use DepositNode to construct the 5 node builder.

The 5 node builder can deposit a node in the construction area and move back to the end of the line by having each node rotate around each other. It is then able to

take another node from the horizontal line by positioning itself two node spaces away from the end of the line and rotating the last node such that it is connected to the seed.

We are therefore able to follow a procedure for constructing vertical lines one node at a time. The construction proceeds for each side of L in phases $0 \leq i \leq |L|$, where phase i corresponds to the construction of the column above the node L_i.

The entire process is mirrored for the bottom of the shape using DepositNode for the builder on the bottom. The builder on the bottom waits until the builder on the top is finished and then starts lifting nodes from the same side of the line. By moving the other builder slightly it is possible to avoid the situation where it disconnects from the line.

Finally, one of the builders places the nodes of the other builder, and is then discarded, leading to a waste of 1 node. By Lemma 3.7, the whole process is completed in $O(n^2)$ time. □

Theorem 3.9. *A nice shape in the class M_n can be transformed to any given nice shape from M_n using a 4-node seed in $O(n^2)$ time.*

Proof. The transformation can be made reversible by assuming that the 4 nodes which are discarded at the end of the transformation constitute a 4 node seed for transforming the nice shape into a line. We can then construct a line of length n by following the process in reverse, and from there construct a nice shape of size n. □

3.2.6 Construction of any Nice Shape

We now show how to extend this to the class of all nice shapes. We follow a broadly similar procedure to the one in Theorem 3.8. The key difference is that we first create the *foundation*, a layer of nodes above and below the horizontal line. We place a node at the start of every vertical line which starts with the same node colour that the previous vertical line built would end with. We then proceed as normal. First we prove that the foundation is sufficient for constructing any colour-consistent nice shape. Then we prove that the 5 node builder is capable of crossing the foundation to deposit nodes.

Lemma 3.10. *For any nice shape constructed from a line, for all lines perpendicular to L with an odd number of nodes there is at most one line which cannot be paired with another line which ends in the other colour.*

Proof. We have the initial line which is either odd or even. We can move nodes out in pairs to build lines. It is possible to build lines which are odd by splitting a pair and distributing its nodes between two odd lines. Such lines can therefore be paired.

However, there are two ways that an extra odd line can be created. First, when the horizontal line is odd, we can support one odd vertical line by extracting the extra node. Second, when the horizontal line is even, we can also split a pair with the horizontal,

making it odd. If both are attempted at the same time the resulting lines will end in different colours and therefore can be paired. As a result, at most one line which is odd cannot be paired. □

Lemma 3.11. *Any* 5 *node builder which is constructing lines can cross the foundation to do so.*

Proof. When moving a builder carrying a node across the foundation, there are 3 scenarios the builder can encounter.

In the first scenario, there is a node in (x, y) which is the same colour as the node being carried in $(x-2, y)$. In this case, the builder must deposit the node in $(x-4, y)$ and collect the node it has encountered. Then, when the builder is returning without carrying a node, it must shift the node it deposited from $(x-4, y)$ to (x, y).

In the second scenario, the node at (x, y) is a different colour and the cell $(x+1, y)$ is empty. For this scenario, the node which is being carried rotates into $(x+1, y)$. Then the builder's nodes rotate around each other to be above $(x-1, y-1)$ and $(x-2, y-1)$. Then the top two nodes in $(x-1, y+1)$ and $(x-2, y+1)$ rotate around each other such that the node in $(x+1, y)$ is the node being carried by the 5 node builder.

In the third scenario, there is a series of nodes beginning with the node (x, y), with alternating colours blocking the builder. In this case, we first identify the node n which is the node furthest right in the series with the same colour as the node the builder is carrying. Then the top two nodes of the builder in $(x-2, y+1)$ and $(x-3, y+1)$ rotate until they are positioned such that they form a 5 node builder with n.

Each of these processes are depicted in Figure 3.8.

Any foundation must consist of any of these three scenarios arranged in a sequence. Therefore, by following the correct process in the scenario the builder crosses the foundation and places a node of the correct colour. Then while returning any nodes deposited can be shifted, creating a new foundation which is equivalent to the original. □

We are now in the position to prove our main result, that it is possible to construct any nice shape from any other nice shape using a seed of size 4. Let N_n be the subclass of nice shapes which is colour consistent to a line of length n.

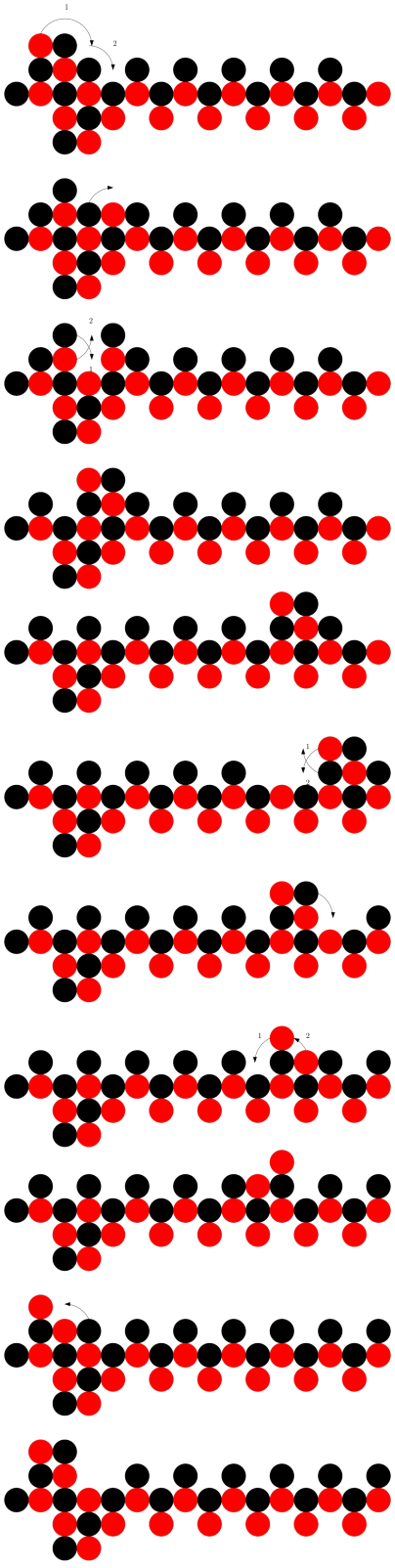
Theorem 3.12. *A line of length n can be transformed to any given nice shape N_{n-1} using a 3-node seed in $O(n^2)$ time.*

Proof. The initial steps of the procedure are as in Theorem 3.8. When we have created both builders, we then create the foundation by placing each node in the foundation from right to left. We alternate between the builders as necessary. By Lemma 3.10, we know that the scenario where the colours we need to place do not match what is available will never occur. By Lemma 3.11, we know that the existence of the foundation

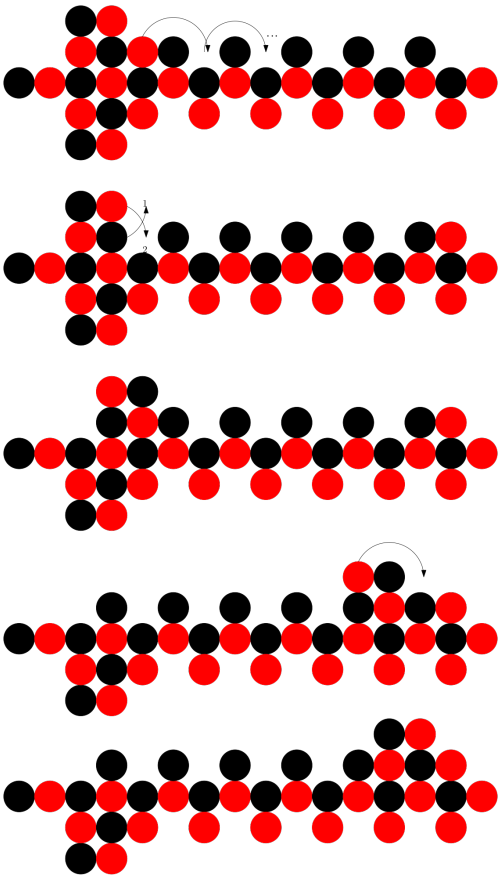
does not impede construction. We are then able to follow a procedure for constructing vertical lines as before. Finally, the last builder is discarded as before. □

Theorem 3.13. *A nice shape of n nodes can be transformed to any given nice shape N_n using a 4-node seed in $O(n^2)$ time.*

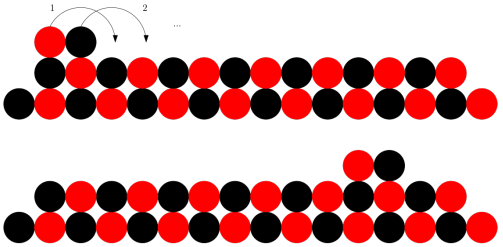
Proof. By Theorem 3.12, we can construct a nice shape from a line using a 3-node seed with 1 node as waste. By reversibility, we can start with a 4-node seed and construct a line of length n. It is then possible to construct another nice shape using the line. □



(A) The case where the colouring is the same.



(B) The case where the colouring is different.



(C) The case where the colours alternate.

FIGURE 3.8: Moving the builder across the foundation.

Chapter 4

Transformations for Orthogonally Convex Shapes

4.1 Introduction

In this chapter, we investigate the same RotC-Transformability problem with the rotation model as the previous chapter, this time with the goal of transforming orthogonally convex shapes, defined in Chapter 2. We again assume the existence of a seed. As in Chapter 3, we move towards a solution which is based on connectivity-preservation and the tighter constraints of rotation-only movement of [31] while aiming to (i) widen the characterization of the class of transformable shapes and (ii) minimise the seed required to trigger those transformations. By achieving these objectives for orthogonally convex shapes, we make further progress towards the ultimate goal of an exact characterisation (possibly universal) for seed-assisted RotC-Transformability.

We study the transformation of shapes of size n with *orthogonal convexity*, the property that for any two nodes u, v in a horizontal or vertical line on the grid, there is no empty cell between u and v, into other shapes of size n with the same property, via the canonical shape of a diagonal line-with-leaves. For our proofs, we partition the diagonal line-with-leaves into a group of components, the main being a series of 2-node columns where each column is offset such that the order of the nodes is equivalent to a line, and two optional components: two 1-node columns on either end of the shape and additional nodes above each column, making them into 3-node columns.

The transformation of orthogonally convex shapes is challenging because of the high level of structural variety, especially compared to nice shapes. While the latter can be represented as a line with other lines of arbitrary length perpendicular to it, the former is intuitively understood as a shape divided into quarters, with the perimeter of the shape represented as a series of alternating horizontal and vertical lines of arbitrary length, following a fixed pattern for direction depending on the quarter and enclosing

an area filled with nodes, as an empty cell would violate orthogonal convexity. Any transformation algorithm must deal with the potential complexity of the perimeter, as well as the fact that the transformation process must make the shape which is being transformed into a shape which violates orthogonal complexity, with the result that the shape is even more complex and difficult to transform algorithmically.

We show that transforming an orthogonally convex shape of n nodes into a diagonal line-with-leaves is possible and can be achieved by $O(n^2)$ moves using a 3-node seed. This bound on the number of moves is optimal for the considered class, due to a matching lower bound from [31] on the distance between a line and a staircase, both of which are orthogonally convex shapes. A seed is necessary due to the existence of blocked orthogonally convex shapes, an example being a rhombus. As shown in Chapter 2, any seed with less than 3 nodes is incapable of non-trivial transformation of a line of nodes. Since a line of nodes is orthogonally convex, the 3-node seed employed here is minimal.

The class of orthogonally convex shapes cannot easily be compared to the class of nice shapes. A diagonal line of nodes in the form of a staircase belongs to the former but not the latter. Any nice shape containing a gap between two of its columns is not an orthogonally convex shape. Finally, there are shapes like a square of nodes which belong to both classes. Nevertheless, the nice shapes that are not orthogonally convex have turned out to be much easier to handle than the orthogonally convex shapes that are not nice. We hope that the methods we had to develop in order to deal with the latter class of shapes, will bring us one step closer to an exact characterisation of connectivity-preserving transformations by rotation.

In Section 4.2, we present preliminary work related to shape generation and elimination sequences which leads into our algorithm. In Section 4.3, we provide our algorithm for the construction of the diagonal line-with-leaves which, through reversibility, can be used to construct other orthogonally convex shapes and give time bounds for it.

4.2 Preliminaries

Recall that in Chapter 2 we defined shape generation and shape elimination sequences. We now build on these definitions with some additional preliminaries.

Let S be an extended staircase of n nodes. An *extended staircase generation sequence* $\sigma = (u_1, u_2, \ldots, u_n)$ of S is a generation sequence of S which consists of q sub-sequences $\sigma = \sigma_1\sigma_2 \ldots \sigma_q$, where each σ_i contains the nodes of the column C_i of S, ordered such that they do not violate the properties of a shape generation sequence. A *diagonal line-with-leaves generation sequence* is an *extended staircase generation sequence* where the repository of the constructed extended staircase is $\emptyset$. See Figure 4.1 for an example.

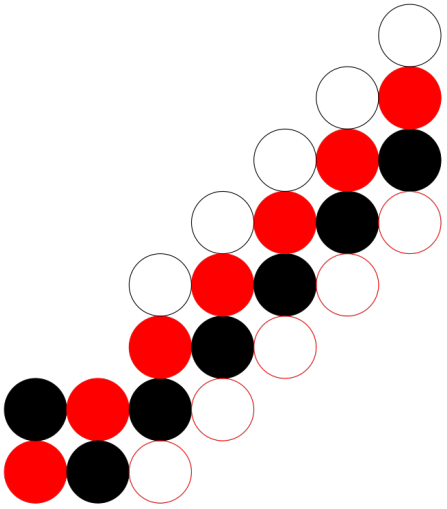


FIGURE 4.1: An example of an extended staircase. The hollow circles represent the black and red repositories, spaces which can be used to hold black and red nodes during construction.

Lemma 4.1. *Every connected orthogonally convex shape S has a row (and column) elimination sequence σ.*

Proof. Let R_1 be the bottom-most row of S, $u_1, u_2, \ldots, u_k$ being the line formed by row R_1. It is sufficient to prove that there is an elimination sequence σ_1 of R_1, as this can then be applied repeatedly to each subsequent bottom-most row R_i, $2 \leq i \leq p$, until S becomes empty, σ then being obtained by $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_p$.

If there is a single node u_j in R_1 which is adjacent to a node v in R_2, then, if $2 \leq j \leq k-1$, $\sigma_1 = (u_1, \ldots, u_j, u_k, \ldots, u_{j+1})$ is an elimination sequence of R_1 and, if $j = 1$ or $j = k$ the same holds for $\sigma_1 = (u_k, \ldots, u_1)$ and $\sigma_1 = (u_1, \ldots, u_k)$, respectively. This holds because, in all these cases, only removing u_{j+1} before the last step in the sequence could disconnect the shape, thus, connectivity is preserved. Moreover, orthogonal convexity is not violated by any removal as this would contradict either the assumption that R_1 is bottom-most or the fact that nodes are only removed from the current endpoints of R_1.

Finally, observe that if multiple nodes in R_1 are adjacent to distinct nodes in R_2, then these must necessarily be consecutive, otherwise orthogonal convexity would be violated in R_2. Setting any of those nodes of R_1 as the u_{j+1} of the previous case, will again give elimination sequences of R_1. □

Lemma 4.2. *For any connected orthogonally convex shape S of n nodes, given a row elimination sequence σ of S and a diagonal line-with-leaves generation sequence σ' of a fixed parity which is colour-order preserving w.r.t σ, the maximum imbalance of any prefix of size $m \leq n$ of σ' is at most $O(2m/3)$.*

Proof. Assume w.l.o.g that the parity of σ is black. If S is a diagonal line-with-leaves with the red parity, where each column in S has 3 nodes, then every prefix of σ' of m nodes will have an imbalance of 2 red nodes for every black node for all m nodes, leading to the maximum imbalance of $O(2m/3)$. □

Lemma 4.3. *For any diagonal line-with-leaves generation sequence σ generating a shape S with a fixed parity column by column, for any sub-sequence σ' which is a prefix of σ, the number of non-parity nodes in σ' cannot exceed the number of parity nodes by more than 2.*

Proof. Assume that there is such a σ', constructing a diagonal line-with-leaves S' of q columns $C_1, C_2, \ldots, C_q$. It must be the case that, in the process of constructing S, the shape generation sequence generates the shape constructed by σ'. We assume w.l.o.g. that the parity of σ (and by extension σ') is black. Therefore, each column C_i in S' constructed by σ' must have at least one black node neighbouring every red node to preserve connectivity. Therefore, σ' has two possible locations to store additional red nodes without increasing the number of black nodes: by placing one red node in C_1 and by placing another in C_q. Placing any more red nodes violates the structure of a black parity diagonal line-with-leaves by making the lowest node in any C_i the non-parity colour, and is therefore impossible. □

For the next proof, we ignore the trivial shape of a node surrounded by four other nodes.

Lemma 4.4. *Let S be a connected orthogonally convex shape. Then there is a row (column) elimination sequence of S which has no single-coloured 3-sub-sequence.*

Proof. Assume that every row (column) elimination sequence σ has such a single-coloured 3-sub-sequence $\sigma' = (u_i, u_{i+1}, u_{i+2})$. Assume there is a row R of S such that $u_i, u_{i+1}, u_{i+2} \in R$. Recall that a row elimination sequence for a given row R is of the form $\sigma^1\sigma^2$ resulting from the partitioning of R into two consecutive lines, where at most one can be empty. It follows that σ' cannot be a sub-sequence of σ^1 or σ^2 because each is an alternating sequence of colours. So, σ' must be spanning the switching point from σ^1 to σ^2, sharing a 2-sub-sequence with either the suffix of σ^1 or the prefix of σ^2. But that 2-sub-sequence cannot be single-coloured because each of σ^1 and σ^2 is an alternating sequence of colours.

Next, we consider the situation where σ' spans multiple rows. Note that if S is a series of one node rows, then σ' cannot contain nodes belonging to different rows of S because any row elimination sequence must switch colour to move between rows. If there are two rows R_1 and R_2, then if R_2 is even then we can select the colour by selecting between σ^1 and σ^2. If R_2 is odd, then both sequences can start with the

same colour, but because each alternates there cannot be a 3-sub-sequence unless one is immediately followed by the other. This is only possible if R_2 is a 3 node line and there is a third line R_3 with one node. Because we ignore the trivial shape, there must be an R_4, and by rotating the row elimination sequence we can get a σ without σ'. □

Given an extended staircase S, an *empty slot* is a cell in the cell perimeter of S which can be occupied by a node u such that $T = S \cup \{u\}$ is an extended staircase.

We now present an algorithm (see Algorithm 2), which when given a row elimination sequence σ returns an extended staircase generation sequence σ'. The algorithm, which is stated in more general terms and works for a larger set of bi-coloured sequences, first constructs a prefix of 4-5 nodes (4 plus an optional repository for a black node) and then extends it by placing nodes on the staircase. If this is not possible, then it places nodes in the repository corresponding to the colour of the node.

The following is an informal description of Algorithm 2. By assumption, it expects the first two nodes of the input sequence to form a bicolour pair, the third node to be black, and no single-colour 3-sub-sequence to ever arrive. The algorithm positions the pair vertically and the black to its right at (x_l, y_d). If the fourth node is black, it goes to an optional single-black repository to the left of the pair and the fifth node must then be red. Otherwise the fourth node is red. In both cases, a red will be placed over the (x_l, y_d) black. Thus, the prefix of the shape constructed by the algorithm always consists of two vertical pairs and the possibility of a black stored at the single-black repository to their left. If the next node is a red it will be stored in the first red repository position at (x_l, y_{d-1}). If not, it is a black. In both cases the next black will start a new column to the right and the algorithm has finished the construction of the prefix having reached its invariant configuration. The invariant satisfies the following properties. New columns always start with the placement of a black. The red repository position of that column below the black and the black repository of the previous column are unoccupied at that point. Any nodes that alternate colours keep growing the staircase part of the shape, preserving the above invariant conditions. If two consecutive nodes of the same colour ever arrive, the second of these nodes will be stored to the first available position of the repository corresponding to its colour. This keeps growing a staircase extended with an upper black and a lower red repository. Both repositories are diagonal lines of consecutive nodes attached to the staircase, starting from its bottom left and having no gaps. The current length of the staircase is an upper bound on the length of the red repository and on the length of the black repository plus 1.

The following assumptions are made by Algorithm 2. The third node is always a black node. This is a necessary technical assumption that we shall later ensure is always satisfied by our transformations. Variables N_B, N_R are assumed to be always

set to the current #nodes in the black, red repository, respectively. The single-black repository at $(x_l - 2, y_d)$, not counted in N_B, stores the fourth node if both the third and the fourth node of the sequence are black.

Algorithm 2 ExtendedStaircase(σ)

Input: row elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$
Output: extended staircase generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving w.r.t. σ

N_B, N_R: current #nodes in the black and red repository, respectively

if $c(u_1) = red$ and $c(u_2) = black$ **then** ▷ 1st and 2nd are always a bicolour pair
  $u'_1 = (x_l - 1, y_d)$, $u'_2 = (x_l - 1, y_d + 1)$
else
  $u'_1 = (x_l - 1, y_d - 1)$, $u'_2 = (x_l - 1, y_d)$
$u'_3 = (x_l, y_d)$ ▷ Assumption that 3rd is always black

if $c(u_4) = black$ **then**
  To be stored at the single-black repository
  $u'_5 = (x_l, y_d + 1)$ ▷ 5th must be red
  $i = 6$
else
  $u'_4 = (x_l, y_d + 1)$
  $i = 5$

for all remaining $i \leq n$ **do**
  If first of new column, then $u'_i = (x_r + 1, y_u)$ ▷ this is always black
  if $c(u_i) \neq c(u_i - 1)$ **then**
    if $c(u_i) = black$ **then**
      $u'_i = (x_r + 1, y_u)$
    else
      $u'_i = (x_r, y_u + 1)$
  else
    if $c(u_i) = black$ **then**
      $u'_i = (x_l + N_B, y_d + N_B + 2)$
    else
      $u'_i = (x_l + N_R, y_d + N_R - 1)$

Lemma 4.5. *Let σ be a bicoloured sequence of nodes that fulfills all the following conditions:*

- *The set of the first two nodes in σ is not single-coloured.*
- *The third node of σ is black.*
- *σ does not contain a single-coloured 3-sub-sequence.*

Then there is an extended staircase generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving with respect to σ.

Proof. The sequence σ' is the one obtained by applying Algorithm 2 to σ. The algorithm begins by placing the first 4 or 5 nodes of σ', depending on whether u_3 is red or black respectively. The result is a shape-prefix with 4 nodes, possibly with an extra black in the repository, with 2 empty black slots and 2 empty red slots neighbouring the nodes in (x_l, y_d) and $(x_l, y_d + 1)$. We now begin to follow the loop of Algorithm 2. When we extend the staircase by one node, this creates a new column with two empty slots for the opposite colour, one in the new column and another in the repository of a third column. When we add a node of that colour to the column, we create two new empty slots for the first colour in the same manner. As a result, the number of empty slots in the repositories only rises as the staircase extends. Therefore, the 3 node restriction of the second condition for σ is the minimum necessary for the worst case where we have only 2 empty slots, and the σ' derived from such a σ by the construction algorithm generates an extended staircase as required. □

ExtendedStaircase is an algorithm which creates an extended staircase generation sequence from a row elimination sequence of a connected orthogonally convex shape.

Lemma 4.6. *For an extended staircase generation sequence σ generated by ExtendedStaircase, every shape generated by a prefix of σ is orthogonally convex.*

Proof. Observe that an extended staircase consists of 4 diagonal lines of nodes: the two diagonals of *Stairs*, and the two nodes which connect to and extend them, *BRep* and *RRep*. The construction of *Stairs* never has a gap between nodes as the lines of the algorithm which add nodes to it require the colour of the nodes to alternate and the algorithm alternates between creating a new column and adding another node to it. The diagonal lines *BRep* and *RRep* grow node by node from the first column of *Stairs* to the last. Their sizes are therefore upper bounded by the size of *Stairs*, and there can be no vertical or horizontal gap. □

Lemma 4.7. *For any connected orthogonally convex shape S of n nodes, given a row elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$ of S where the set of the first two nodes in σ is not single-coloured and u_3 is black, there is an extended staircase generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving w.r.t σ and such that, for all $1 \leq i \leq |\sigma|$, $D_i = \{u'_1, u'_2, \ldots, u'_i\}$ is a connected orthogonally convex shape.*

Proof. By Lemma 4.4, σ will not have a single-coloured 3-sub-sequence. Therefore, by our assumption about σ and Lemma 4.5 we have a σ'. We can then place the nodes of σ' as in Algorithm 2. By Lemma 4.6, all prefixes σ'_i of σ' construct an orthogonally convex shape (excluding the black repository), and therefore all D_i are connected orthogonally convex shapes. □

Observation 2. For any connected orthogonally convex shape S of n nodes, if the set of the first two nodes in the row elimination sequence $\sigma = (u_1, u_2, u_3, \ldots, u_n)$ is single-coloured, u_3 is black and there is an empty cell c of the opposite colour in the cell perimeter of S such that if c is occupied by v then $S \cup \{v\}$ is an orthogonally convex shape, then $S \cup \{v\} \setminus u_1$ has a row elimination sequence σ' where the set of the first two nodes in σ' is not single-coloured.

The *anchor node* of the shape S of p rows $R_1, R_2, \ldots, R_p$ is the rightmost node in the row R_p, counting rows from bottom to top. *ExtendedStaircase* is an algorithm which creates an extended staircase generation sequence from a row elimination sequence of a connected orthogonally convex shape.

Lemma 4.8. *Let S be a connected orthogonally convex shape of n nodes divided into p rows $R_1, R_2, \ldots, R_p$, and $\sigma = (u_1, u_2, \ldots, u_n)$ a row elimination sequence from R_1 to R_p of S. If the bottom node of the first two nodes placed by ExtendedStaircase is fixed to $(x_c, y_c + 1)$, where (x_c, y_c) are the co-ordinates of the anchor node of S, the shape $T_i = ExtendedStaircase(\sigma_i)$, where $\sigma_i = (u_1, u_2, \ldots, u_i)$, $1 \leq i \leq n$, fulfills the following properties:*

- *$S \cup T_i$ is a connected shape.*
- *$S \cap T = \emptyset$.*
- *excluding the single-black repository, $R_p \cup T_i$ is an orthogonally convex shape.*

Proof. Let $u_1, u_2, \ldots, u_i$ be the nodes in the sequence σ_i. If the first node is black, Algorithm 2 places a node in $(x_l - 1, y_d - 1)$, otherwise it places it in $(x_l - 1, y_d)$. By Lemma 4.6, all σ_i generate an orthogonally convex shape, so T_i cannot be a disconnected shape. Therefore, the shape $S \cup T_i$ is connected. In addition, the co-ordinates $(x_l - 1, y_d - 1)$ and $(x_l - 1, y_d)$ represent the two potential bottom-left corners of the shape T. Therefore, there can be no overlap (i.e. placement of nodes in occupied cells) as the existence of a node of S in the space T is constructed in would contradict the definition of an anchor node. In addition, the cell $(x_l - 2, y_d)$ (the single-black repository) is always empty as a node in that cell would have the y co-ordinate y_d, which is above the anchor node at $y_d - 2$ or $y_d - 1$, violating the definition of the anchor node. Finally, since the nodes u_1 and u_2 construct a column, and every node $u_3, \ldots, u_n$ (excluding the single-black repository) is necessarily to the right of this column, there cannot be a violation of orthogonal convexity with the row R_p. □

Lemma 4.9. *For any extended staircase $W \cup T$ of n nodes, where W is the $Stairs$, $T \subseteq \{BRep \cup RRep\}$ and $k = |T|$, given a shape elimination sequence $\sigma = (u_1, u_2, \ldots, u_k)$ of T, there is a diagonal line-with-leaves generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_k)$ which*

is colour-order preserving w.r.t σ *and such that, for all* $1 \leq i \leq |\sigma|$, $D_i = W \cup \{u'_1, u'_2, \ldots, u'_i\}$ *is a connected orthogonally convex shape.*

Proof. We use a shape elimination sequence σ of T which alternates between taking nodes from the black repository $BRep$ and the red repository $RRep$. It does this until only one repository remains. We can then use σ' to place the nodes of D as in the for loop of Algorithm 2, effectively extending W. If we maximise the size of σ then the resulting D_k, a $Stairs$ with only one repository, is equivalent to a diagonal line-with-leaves. By Lemma 4.6, all prefixes D_i generated by σ' are connected orthogonally convex shapes (excluding the black repository). □

4.3 The Transformation

In this section, we present the transformation of *orthogonally convex* shapes, via an algorithm (Algorithm 3) for constructing a diagonal line-with-leaves from any orthogonally convex shape S. For the first step of the algorithm, we generate a 6-robot from the seed and the shape, which we then use to transport nodes. By using a row elimination sequence of S and an extended staircase generation sequence, we convert the initial shape S into an extended staircase. We then use appropriate elimination and generation sequences focused on the repositories of the extended staircase, to convert the latter into a diagonal line-with-leaves. Given any two colour-consistent orthogonally convex shapes A and B and their diagonal line-with-leaves D, our algorithm can be used to transform both A into D and B into D and, thus, A into B, by reversing the latter transformation. This transformation applies to all orthogonally convex shapes with 3 nodes. A 2-node shape can trivially transform by rotating one node around the other and a 1-node shape cannot transform at all.

Our transformations rely on the use of a *k-robot*, a shape with k nodes which is responsible for transporting nodes. The k-robot *extracts* a node u if it is positioned such that u rotates around a node of the robot and the result is a $k+1$-robot where u is the load of the robot. The $k+1$-robot *places* its load in the cell c if it is positioned such that the load rotates into c and the result is a k-robot.

Algorithm 3 OConvexToDLL(S, M)

Input: shape $S \cup M$, where S is a connected orthogonally convex shape of n nodes and M is a 3-node seed on the cell perimeter of S, row elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$ of S, extended staircase generation sequence of $W \cup T = \sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving w.r.t. σ, shape elimination sequence $\sigma = (u_1, u_2, \ldots, u_{|T|})$ of T, shape generation sequence of $X = \sigma' = (u'_1, u'_2, \ldots, u'_{|T|})$ which is colour-order preserving w.r.t. σ

Output: shape $G = W \cup X \cup M$, where G is a diagonal line-with-leaves and M is a connected 3-node shape on the cell perimeter of S.

$R \leftarrow$ GenerateRobot(S, M)
$\sigma \leftarrow$ rowEliminationSequence(S)
$\sigma' \leftarrow$ ExtendedStaircase(σ)
$W \cup T \leftarrow$ OConvexToExtStaircase(S, R, σ, σ')
$\sigma \leftarrow$ repsEliminationSequence($W \cup T$)
$\sigma' \leftarrow$ stairExtensionSequence($W \cup T$)
$G \leftarrow$ ExtStaircaseToDLL($W \cup T, R, \sigma, \sigma'$)
TerminateRobot(G, R)

Algorithm 4 OConvexToExtStaircase(S, R, σ, σ')

Input: shape $S \cup R$, where S is a connected orthogonally convex shape of n nodes and R is a 6-node robot on the cell perimeter of S, row elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$ of S, extended staircase generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving w.r.t. σ

Output: shape $T \cup R$, where T is the extended staircase generated by σ'

for all $1 \leq i \leq n$ **do**
  $source \leftarrow \sigma_i$
  $dest \leftarrow \sigma'_i$
  while R cannot extract source **do**
    if R can climb **then**
      $Climb(R)$
    else
      $Slide(R)$
  $Extract(R, source)$
  while R cannot place its load in dest **do**
    if R can climb **then**
      $Climb(R)$
    else
      $Slide(R)$
  $Place(R, dest)$

Algorithm 5 ExtStaircaseToDLL(W, R, σ, σ')

Input: extended staircase $W = Stairs \cup \{BRep \cup RRep\}$ and a 6-robot R on its cell perimeter, shape elimination sequence $\sigma = (u_1, u_2, \ldots, u_{|T|})$ of $T \subseteq \{BRep \cup RRep\}$, shape generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_{|T|})$ which is colour-order preserving w.r.t. σ

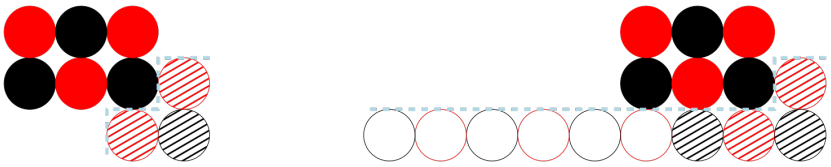
Output: shape $Stairs' \cup R'$, where $Stairs' \setminus Stairs$ is an extension of $Stairs$ generated by σ' and R' is a 6-robot which is colour-consistent with R.

```
for all 1 ≤ i ≤ |T| do
    source ← u_i
    dest ← u'_i
    while R not at source do
        if R can climb then
            ClimbTowards(R, source)
        else
            SlideTowards(R, source)
    Extract(R, source)
    while R not at dest do
        if R can climb then
            ClimbTowards(R, dest)
        else
            SlideTowards(R, dest)
    Place(R, dest)
```

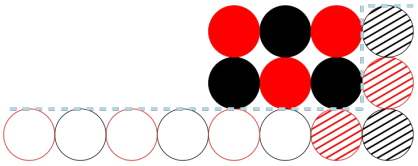
4.3.1 Robot Traversal Capabilities

6-Robot Movement

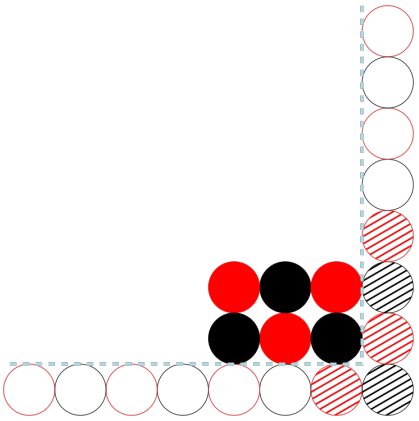
We first show that for all S in the family of orthogonally convex shapes, a connected 6-robot is capable of traversing the perimeter of S. We prove this by first providing a series of scenarios which we call *corners*, where we show that the 6-robot is capable of making progress past the obstacle that the corner represents. We then use Proposition 2.15 to show that the perimeter of any S is necessarily made up of a sequence of such corners, and therefore the 6-robot is capable of traversing it.



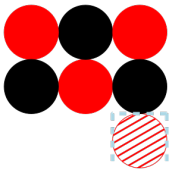
(A) The height 1 cases, with widths 1 and 2+.



(B) The height 2 case.



(C) The height 3+ case.



(D) An extreme version of the partial quadrant, when a quadrant consists of a single line.

FIGURE 4.2: The five cases considered in the proof, four corner cases and one edge case where a section of the perimeter does not correspond to a corner case due to its structure. Striped circles represent the nodes on the exterior of the shape. Hollow circles represent potential space for additional nodes for corner scenarios which are not in this set (due e.g. to having longer horizontal/vertical lines).

We define progress as the movement of the 6-robot upwards and to the right of its starting position, i.e. any change in the position of the shape such that the shape is in the same $a \times b$ formation but the co-ordinates of all of the nodes have increased. This is equivalent to being able to traverse the relevant section of a perimeter. Our goal is to show that attaining the maximum progress (i.e. the movement which maximises the increase in the co-ordinates while preserving the formation) for each corner is possible. Since we can construct a series of corners where every corner follows from the point of maximum progress of the previous corner, it follows that for such a series we can make progress indefinitely. By rotating the robot and the quadrant as necessary, we can make the same argument for progress in any direction, which is equivalent to being

able to traverse a perimeter indefinitely, provided we also show that the perimeter is necessarily made up of such corners.

We begin by considering the *up-right* quadrant, that is any cells which neighbour the section of the perimeter defined by the regular expression $d_1(d_1 \mid d_2)^* d_2\ (d_2 \mid d_3)^* d_3$, where d_1, d_2 and d_3 are *up*, *right* and *down* respectively, as our base case.

Let $\mathcal{C}$ be a set of orthogonally convex shapes, where each shape is a corner scenario for the $up - right$ quadrant, depicted in Figure 4.4. Given a corner-shape scenario $C \in \mathcal{C}$ consisting of a horizontal line $(x_l, y_d), (x_l + 1, y_d), \ldots, (x_r, y_d)$ and a vertical line $(x_r, y_d), (x_r, y_d + 1), \ldots, (x_r, y_u)$, as depicted in Figure 4.4, we define its *width* $w(C) = |x_r - x_l|$, i.e., equal to the length of its horizontal line, and its *height* $h(C) = |y_u - y_d|$, i.e., equal to the length of its vertical line, excluding in both cases the corner node (x_r, y_d).

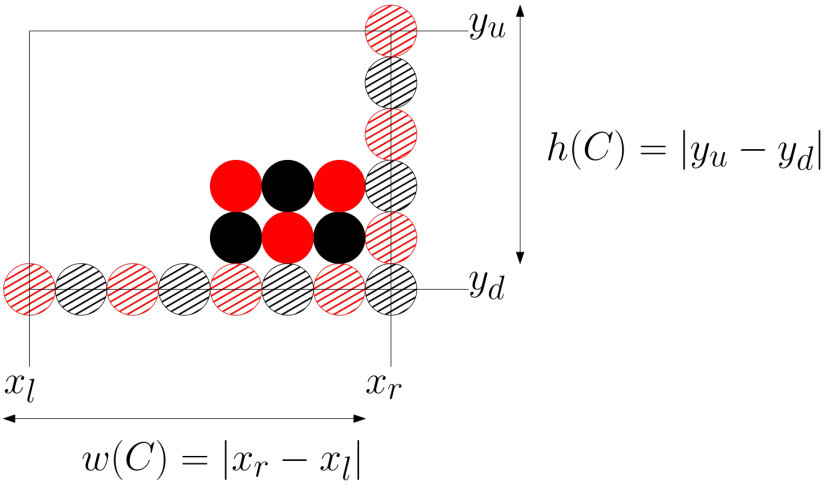
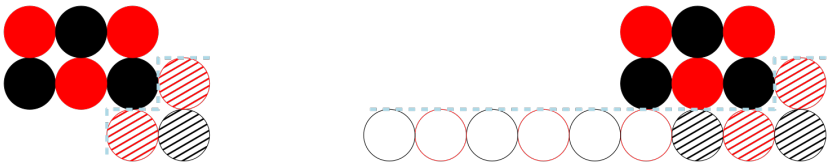
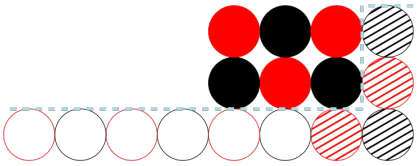


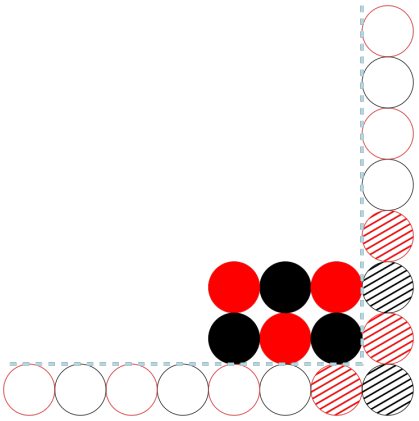
FIGURE 4.3: A visual representation of the variables we use in our proof.



(A) The height 1 cases, with widths 1 and 2+.



(B) The height 2 case.



(C) The height 3+ case.

FIGURE 4.4: The four basic corner scenarios of $\mathcal{C}$.

Lemma 4.10. *For any orthogonally convex shape S, the extended external surface defined by the regular expression $d_1(d_1 \mid d_2)^* d_2\ (d_2 \mid d_3)^* d_3$ of the shape can be divided into a series of shapes $S_0, S_1, \ldots$, where all $S_i \in \mathcal{C}$.*

Proof. By Proposition 1, there is the section of the perimeter of orthogonally convex shapes which is defined by the regular expression $d_1(d_1 \mid d_2)^* d_2\ (d_2 \mid d_3)^* d_3$, where d_1, d_2 and d_3 are *up*, *right* and *down* respectively. This section of the perimeter forms a "quadrant" where all movement is in the up and right directions, terminated by the first d_3, as can be seen in the examples in 2.10, the example shapes with labelled perimeters from Chapter 2. By the regular expression, the nodes on the perimeter must necessarily form alternating horizontal and vertical lines. We can therefore divide this section of the perimeter into a series of subsections, where in each subsection we have a horizontal line which connects to a vertical line via the right-most node on the horizontal line. The cases in Figure 4.4 cover all potential widths and heights where this vertical line positioning constraint holds. This even holds for the edge cases where a vertical ending at (x_r, y_u) is immediately followed by another vertical starting at $(x_r + 1, y_u)$, provided we allow (x_r, y_u) to act both as (x_r, y_u) and (x_l, y_d) for each case respectively. Therefore, they cover all potential cases in the quadrant, and since the quadrant is made up of these cases, it covers the extended external surface of the whole quadrant. □

Given that the quadrant is made up of cases from $\mathcal{C}$, if the 6-robot is able to move from one vertical to another for all $S_i \in \mathcal{C}$, it is able to do so for any up-right quadrant of the perimeter until it runs into the d_3 line. We now show that this movement is possible, first for this quadrant and later for all four quadrants.

Lemma 4.11. *For all shapes $C \in \mathcal{C}$, if a 2×3 shape (the 6-robot) is placed in the cells $(x_l - 2, y_d + 1), (x_l - 1, y_d + 1), (x_l, y_d + 1), (x_l - 2, y_d + 2), (x_l - 1, y_d + 2), (x_l, y_d + 2)$, it is capable of translating itself to $(x_r - 2, y_u + 1), (x_r - 1, y_u + 1), (x_r, y_u + 1), (x_r - 2, y_u + 2), (x_r - 1, y_u + 2), (x_r, y_u + 2)$.*

Proof. For our proof strategy, we present a series of motions which for all shapes $C \in \mathcal{C}$ lead the 6-robot from the leftmost node of the horizontal to the topmost node of the vertical. We group some of these motions into high-level motions (i.e. moving the whole 6-robot by moving individual nodes). We begin by noting that we can perform repetitive motions to traverse a horizontal or vertical line to the end. Our first motion is *sliding*, depicted in Figure 4.5, where pairs of nodes rotate around each other to slide across a line. By reorienting the shape, the 6-robot and its movement vertically, it follows that the 6-robot can slide up vertical lines as well. The second action is a special version of the slide depicted in Figure 4.6. This slide is slower but allows the object to preserve connectivity in the situation where only one or two nodes are connected to the line. We are therefore already able to claim that moving across and onto lines is possible. What remains is the intersection of horizontal and vertical lines.

Our first is the case where the 6-robot lies on a horizontal of arbitrary width, and attempts to climb a vertical of height 3. It does by following the series of rotations in Figure 4.7. The result is that the robot lies on top of the vertical line, and is therefore able to use the slide and/or special slide movements (if necessary) to move across the horizontal line at the top (not depicted) to the next vertical. This climbing procedure can be performed no matter how long the first horizontal (i.e. the one the robot lies on initially) is. In addition, there is a section of the movement which puts the seed in the position to slide vertically (see Figure 4.8), allowing a modified version the same procedure to climb verticals of arbitrary height.

Finally, there are the solutions for the cases where the height is 2 (Figure 4.9) and where the height is 1 (Figure 4.10 and 4.11). Note that unlike the former movement, the latter movements vary depending on the width (2+ and 1 respectively). □

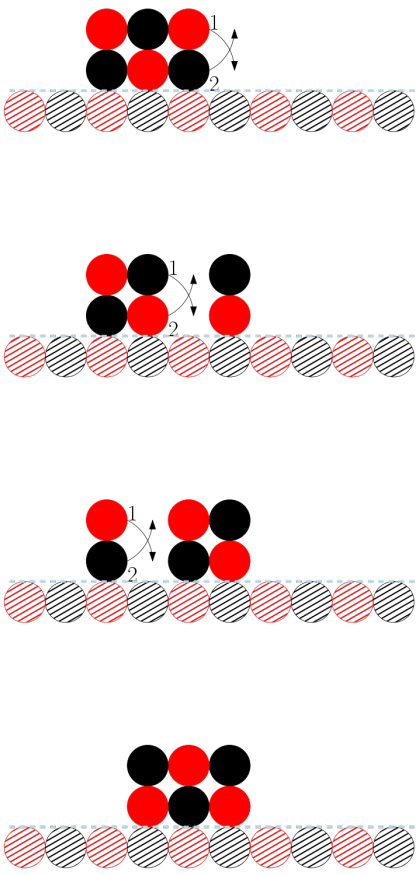


FIGURE 4.5: Sliding across a horizontal line.

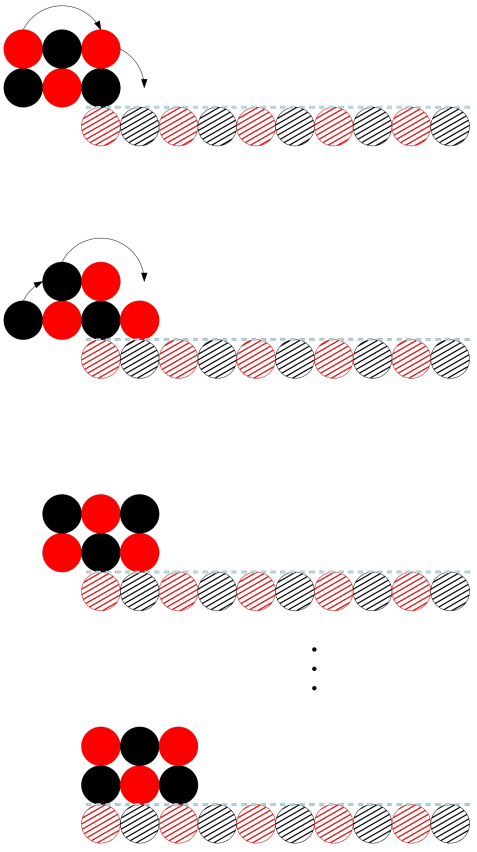


FIGURE 4.6: Sliding onto a horizontal line. The steps are repeated after the third configuration to reach the fourth configuration.

Theorem 4.12. *For any orthogonally convex shape S, a 6-robot is capable of traversing the perimeter of S.*

Proof. By Lemma 4.11 we have shown for the up-right quadrant firstly that it is possible to slide across a horizontal of arbitrary width no matter the robot's initial position, that it is possible to climb a height 3 vertical, that part of this movement can be repeated indefinitely to climb verticals of arbitrary height, that special movements exist for smaller verticals and that all of this is possible no matter how long the horizontal line the object lies on is.

By rotating the robot and the quadrant as necessary, we are able to replicate our movements for all other quadrants: $d_4(d_4 \mid d_1)^*$ (the left-up quadrant) $d_2(d_2 \mid d_3)^*$ (the right-down quadrant) and $d_3(d_3 \mid d_4)^*$ (the left-down quadrant). All that remains is the transition between the quadrants.

There are two cases. In the first case, the next quadrant consists of multiple lines. In this case, when the line signifying the end of the current quadrant is met it is sufficient to begin movements appropriate to travelling in the next quadrant. However, there is

an edge case where a quadrant consists of a single line. In this case, a unique movement is necessary (see Figure 4.12) to transfer the 6-robot onto the line. These movements are then followed by special slides to put the object into position for the next quadrant. Naturally, these transformations are reversible and can be mirrored as well. We are therefore able to deal with any quadrant transition, even rotating the 6-robot around a single node.

Therefore, because we can move through any variant of all quadrants and transition between them, a 6-robot can traverse the perimeter of any orthogonally convex shape. □

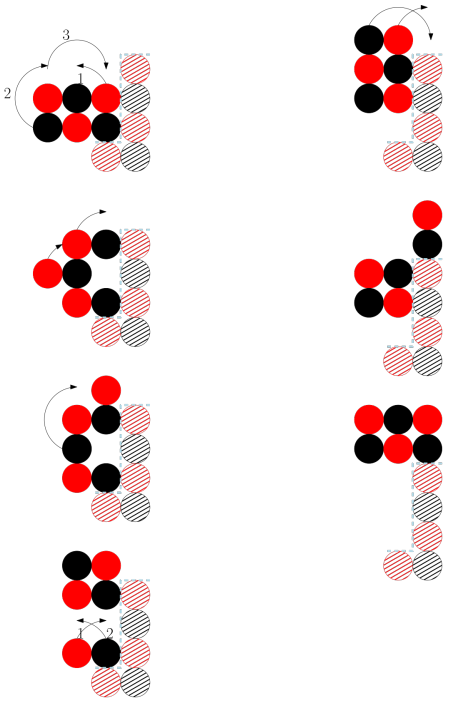


FIGURE 4.7: Climbing a height 3 vertical with a width 1 horizontal. All figures are read as pairs of columns, top-down.

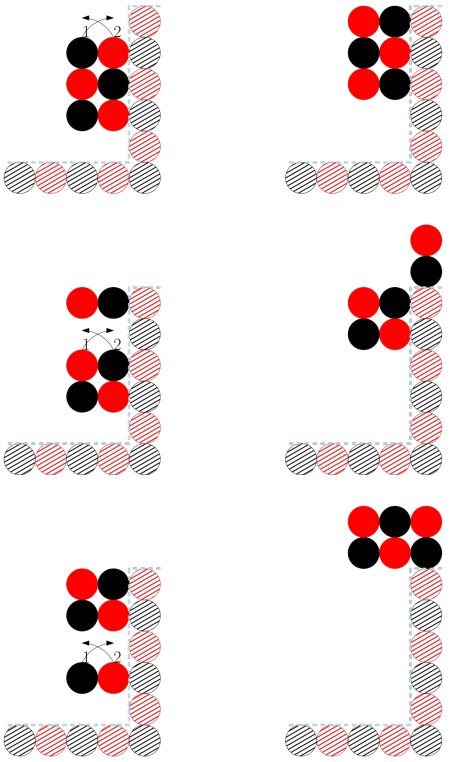


FIGURE 4.8: Climbing a vertical of arbitrary height with an arbitrary width horizontal. The process starts as in Figure 4.7, and the upwards slide in the first column of snapshots can be repeated for as long as necessary to climb the wall. This corresponds to case (c) of Figure 4.4.

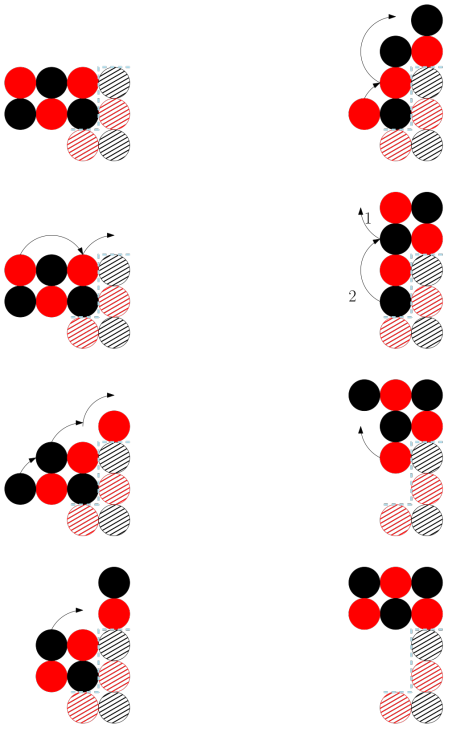


FIGURE 4.9: Climbing a height 2 vertical with an arbitrary width horizontal. This corresponds to case (b) of Figure 4.4.

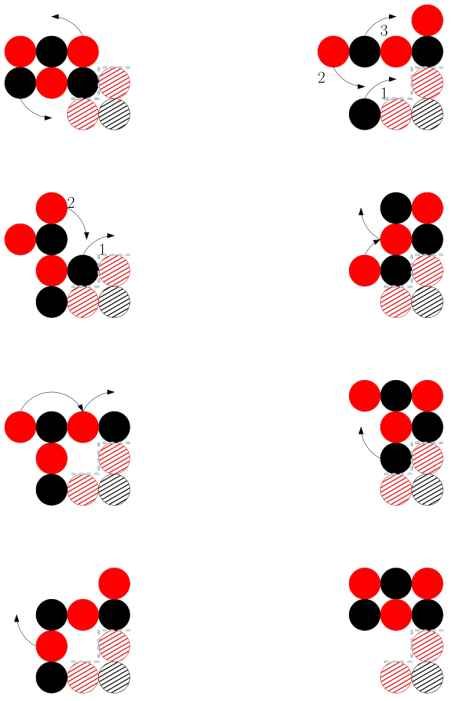


FIGURE 4.10: Climbing a height 1 vertical with a width 1 horizontal. This corresponds to the first (a) case of Figure 4.4.

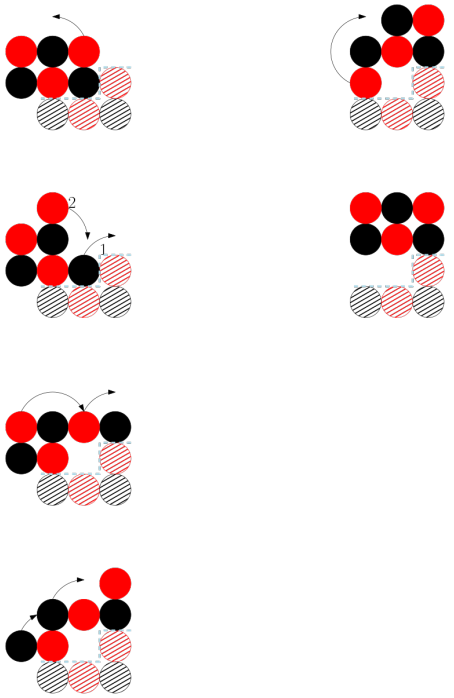


FIGURE 4.11: Climbing a height 1 vertical with a width 2+ horizontal. This corresponds to the second (a) case of Figure 4.4.

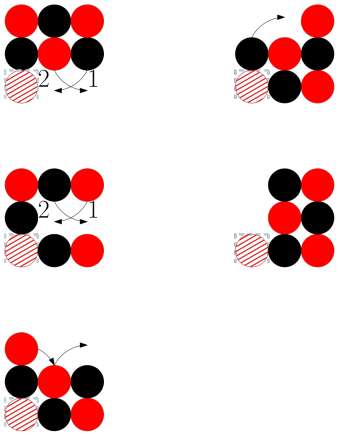


FIGURE 4.12: Movement into a new quadrant consisting of a line of length 1.

7-Robot Movement

We consider once again the *up-right* quadrant, and generalise to other quadrants later. We say a cell $c = (x, y)$ is *behind* the robot if x is smaller than the x-co-ordinate of every node in the robot.

The *load* of a 7-robot S is any node u such that $S \setminus \{u\}$ is a 2×3 shape. The *position* of the robot is an offset of the y axis (see y_d from Figure 4.3) for the purpose of the initial positioning of the 7-robot. For our transformations, we maintain the invariant that the 7-robot, after any of its high-level movements, will return to the structure of a 2×3 shape with a load. For this invariant, we assume that the load is always *behind* the 2×3 shape (while remaining connected). We will show in the proof why the situation where the load is positioned differently does not need to be considered. We therefore use $(x, y|y')$ to refer to the co-ordinates of the two cells (x, y) and (x, y') *behind* the robot which can contain the load, keeping it attached to the robot while the latter is a 2×3 shape. The *colouring* of a 7-robot is *good* if the load is in the higher of the two possible positions, and *bad* if it is in the lower position. Bad colouring usually means the resulting transformations are more difficult.

Lemma 4.13. *For all shapes $C \in \mathcal{C}$, if a 2×3 shape with a load (the 7-robot) is placed in the cells $(x_l - 3, y_d + 1|y_d + 2), (x_l - 2, y_d + 1), (x_l - 1, y_d + 1), (x_l, y_d + 1), (x_l - 2, y_d + 2), (x_l - 1, y_d + 2), (x_l, y_d + 2)$, it is capable of translating itself to $(x_r - 3, y_u + 1|y_u + 2), (x_r - 2, y_u + 1), (x_r - 1, y_u + 1), (x_r, y_u + 1), (x_r - 2, y_u + 2), (x_r - 1, y_u + 2), (x_r, y_u + 2)$.*

Proof. We present a series of motions which for all shapes $C \in \mathcal{C}$ lead the 7-robot from the leftmost node of the horizontal to the topmost node of the vertical. As in Lemma 4.11, we group some of these motions into high-level motions (i.e. moving the whole 7-robot by moving individual nodes).

Our first motion is *sliding*, depicted in Figures 4.13 and 4.14. The 7-robot can alternate between these two transformations to slide over the horizontal line of C. Alternating between the two is possible because the final configuration of each has the form required by the initial configuration of the other. The second motion is also a version of sliding, called *special sliding*, depicted in Figures 4.15-4.17. The purpose of special sliding is to bring the 7-robot "onto" the horizontal line, when it starts from an extreme position from which the sliding motion does not apply. Figures 4.15 and 4.16 cover the cases where only the bottom-right node of the 7-robot is attached to the horizontal line, while Figure 4.17 the case where there are two points of attachment but the robot colouring is bad. By special sliding, the 7-robot can move onto the horizontal line and sliding can then be used to move it across the horizontal, until its bottom-right node is at $(x_r - 1, y_d + 1)$, i.e., in one of the initial configurations of Figures 4.18 and 4.20 (disregarding the height of the vertical). This covers the horizontal part of C. It remains to be shown that the 7-node seed can then climb up and then onto the vertical

part of C. We do this with a motion which we refer to as *climbing*, which covers a few sets of cases.

For the first set of cases, we consider the situations where the height of the vertical is 1. If the colouring is good, then we can rotate the load above the vertical. This is depicted in Figure 4.18. Note that the load necessarily takes the higher of the two possible cells behind the robot at the start of the translation. If the colouring is bad, then the movements depend on the width of the horizontal, which can be 1 (Figure 4.19), and 2+ (Figure 4.20). Note that these movements always deposit the 7-robot in the position for a special slide, and the load always remains behind the robot.

For the next set of cases, we consider the situations where the height of the vertical is 2. Our first is the case where the colouring is bad. In this case, we follow the rotations of Figure 4.21. When the colouring is good, we have two additional cases. We can follow the rotations of Figure 4.22 and Figure 4.23 to climb up and onto the vertical, respectively.

Finally, we consider the cases where the height of the vertical is at least 3. When the height is exactly 3 and the colouring is bad, we can follow Figure 4.24 to reach the top of the vertical and then Figure 4.23 to climb onto it. In the good colouring case, we can repeat the rotations of Figure 4.22 and Figure 4.25 until we reach the top of the vertical, and then perform the rotations of Figure 4.18 or Figure 4.23, depending on whether the load is in the higher or lower of the two possible positions. We do the same for when the height is over 3 and the colouring is bad, but begin with the Figure 4.24 rotation.

We have thus shown how the 7-robot can climb up and onto verticals of any possible length. Putting everything together, starting from one of the initial positions specified by the lemma statement relative to the horizontal line of C, the 7-robot can use special sliding to move onto the horizontal, followed by sliding to move across it, and finally climbing to move up and onto the vertical, for all possible widths and heights of C. Moreover, the final position of the 7-robot relative to the vertical is as required by the statement. □

Theorem 4.14. *For any orthogonally convex shape S, a 7-robot is capable of traversing the perimeter of S.*

Proof. By Lemma 4.13 we have shown for the up-right quadrant firstly that it is possible to slide across a horizontal of arbitrary width no matter the robot's initial position, that it is possible to climb a height 2 vertical, that part of this movement can be repeated indefinitely to climb verticals of arbitrary height, that special movements exist for the height 1 vertical and that all of this is possible no matter how long the horizontal line the object lies on is, nor whether the 7-robot is red or black.

By rotating the robot and the quadrant as necessary, we are able to replicate our movements for all other quadrants: $d_4(d_4 \mid d_1)^*$ (the left-up quadrant) $d_2(d_2 \mid d_3)^*$ (the right-down quadrant) and $d_3(d_3 \mid d_4)^*$ (the left-down quadrant). All that remains is the transition between the quadrants.

There are two cases. In the first case, the next quadrant consists of multiple lines. In this case, when the line signifying the end of the current quadrant is met it is sufficient to begin movements appropriate to travelling in the next quadrant. However, there is an edge case where a quadrant consists of a single line. In this case, a unique movement is necessary (see Figure 4.26 and Figure 4.27) to transfer the 7-node object onto the line, with the exact movement depending on whether the first node of the line is the same or a different colour from the load. These movements are then followed by special slides to put the object into position for the next quadrant. Naturally, these transformations are reversible and can be mirrored as well. We are therefore able to deal with any quadrant transition, even rotating the 7-node object around a single node.

Therefore, because we can move through any variant of all quadrants and transition between them, a 7-robot can traverse the perimeter of any orthogonally convex shape. □

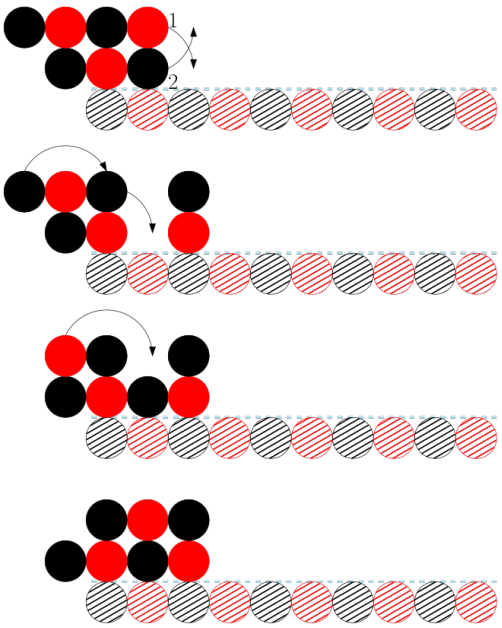


FIGURE 4.13: Sliding across a line with a 7-node robot - case 1

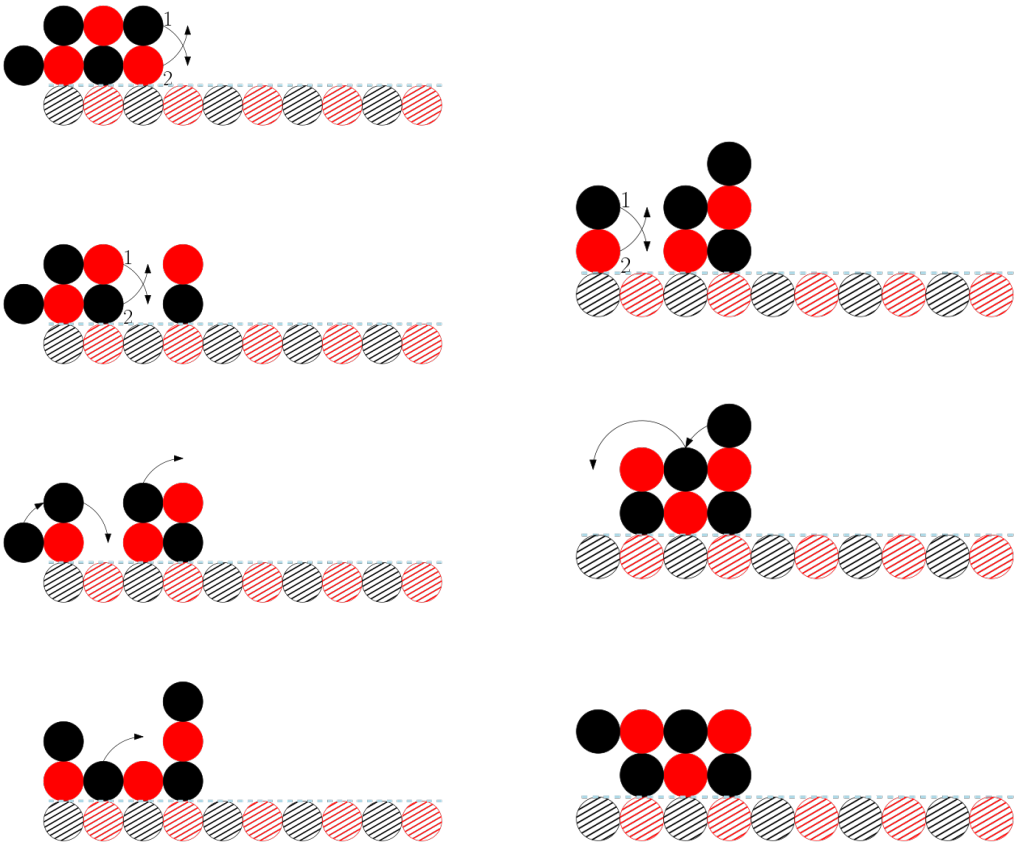


FIGURE 4.14: Sliding across a line with a 7-node robot - case 2

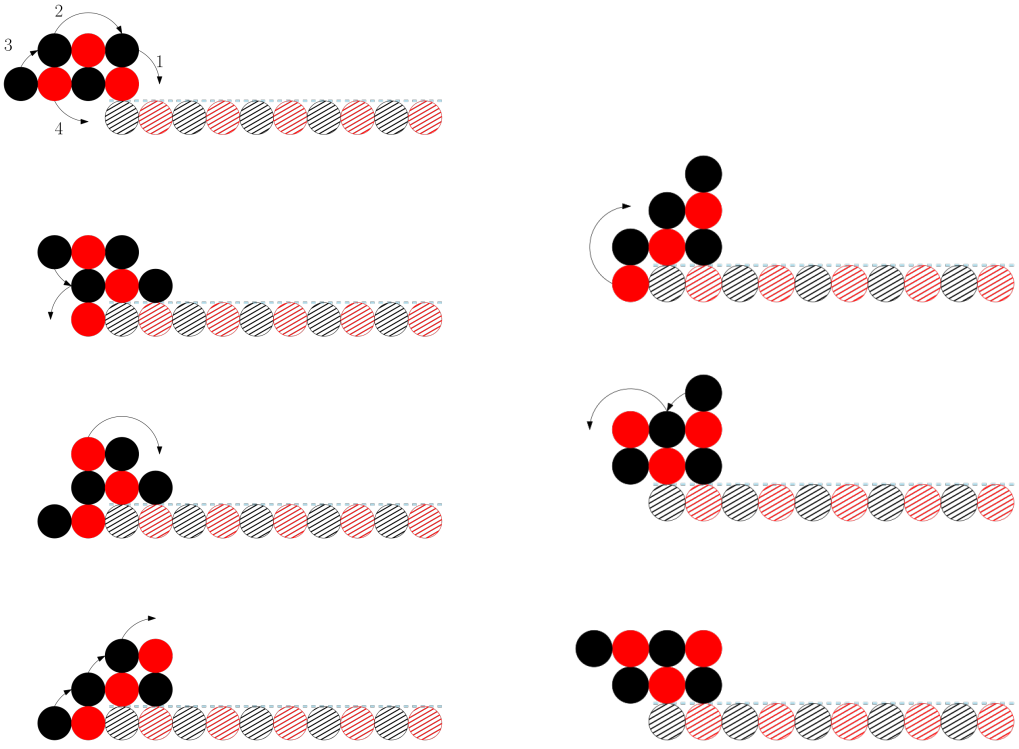


FIGURE 4.15: Sliding on a line with a 7-node robot - case 1

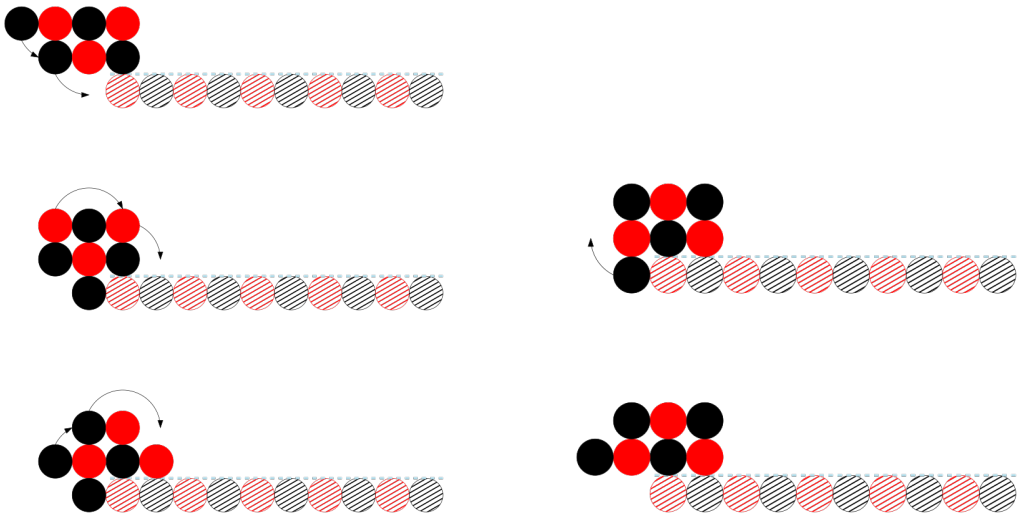


FIGURE 4.16: Sliding on a line with a 7-node robot - case 2

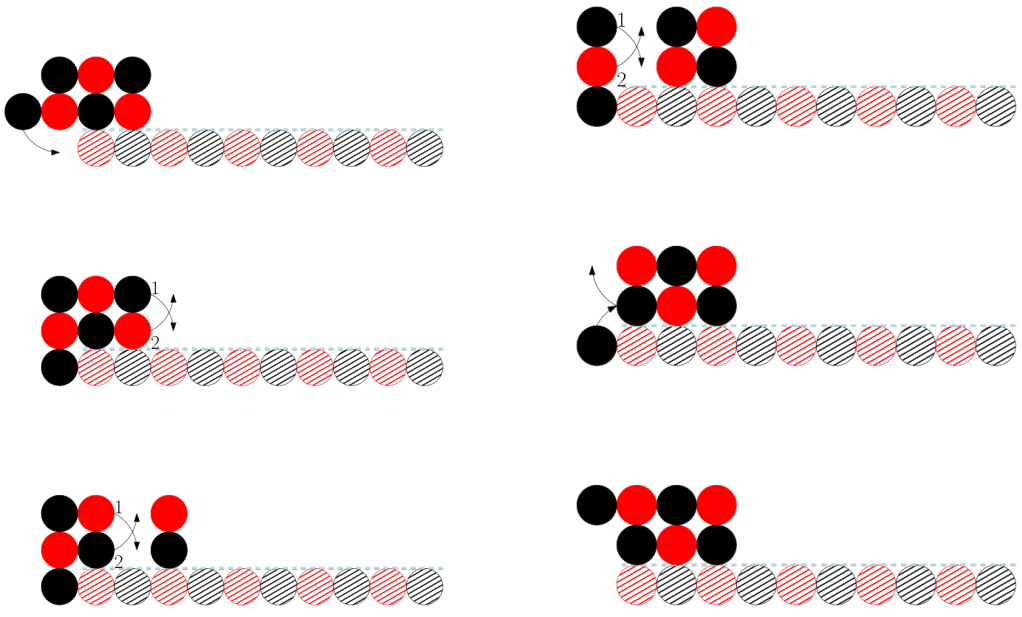


FIGURE 4.17: Sliding on a line with a 7-node robot - case 3. The transformation of Figure 4.13 can be applied for further movement.

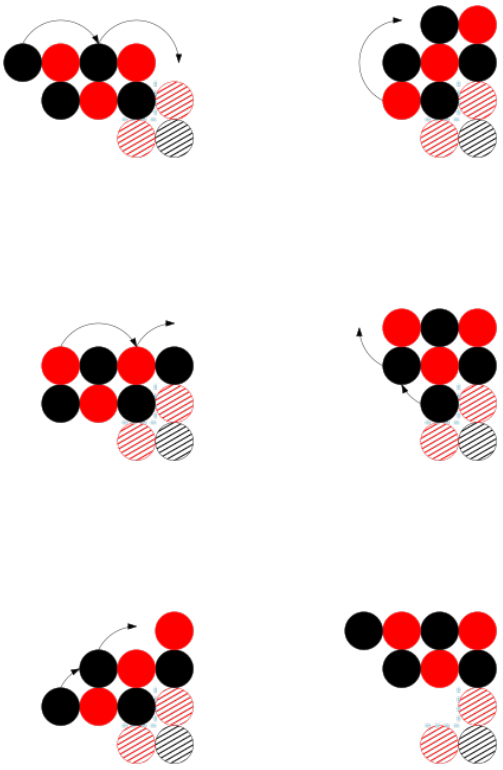


FIGURE 4.18: Climbing on top of a vertical when the load is in the upper cell.

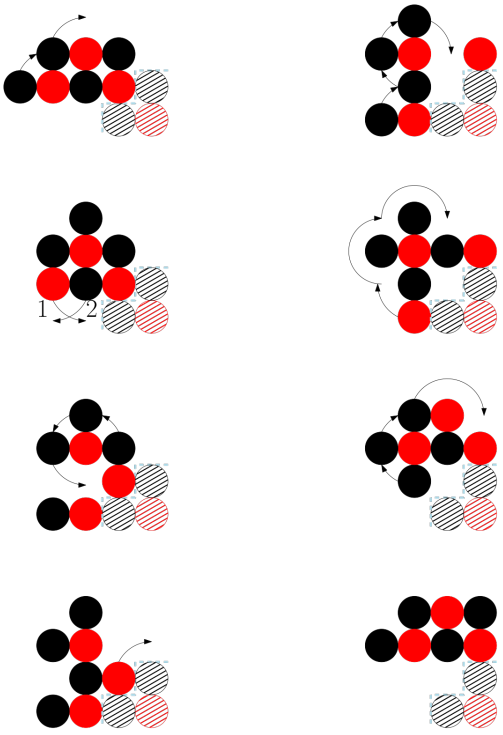


FIGURE 4.19: Climbing a height 1 vertical with a width 1 horizontal and bad colouring

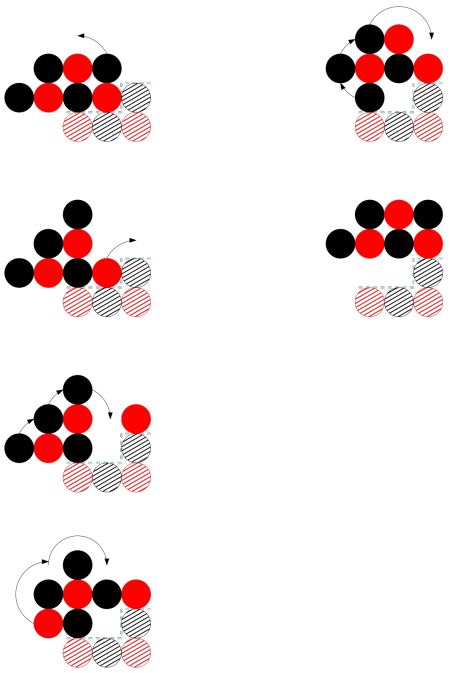


FIGURE 4.20: Climbing a height 1 vertical with a width 2+ horizontal and bad colouring

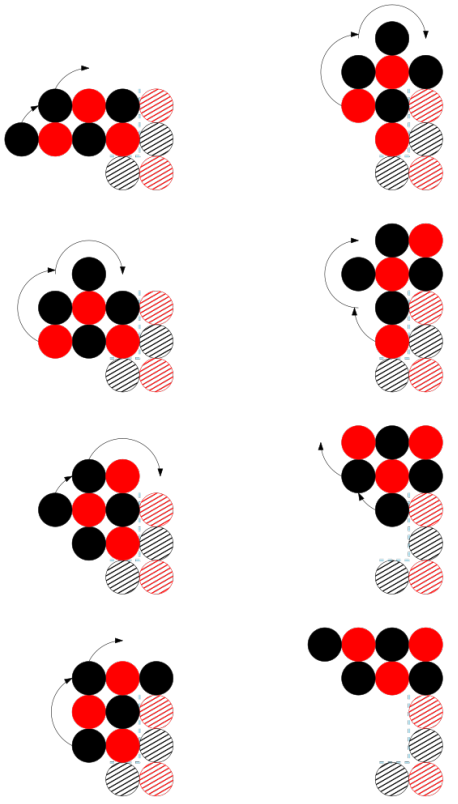


FIGURE 4.21: Climbing on top of a vertical of height 2 from position 0 with bad colouring.

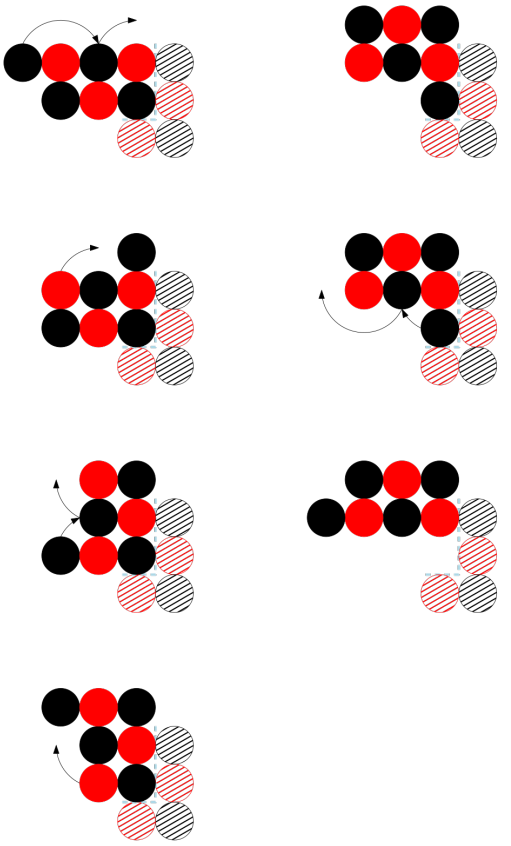


FIGURE 4.22: Climbing a vertical of height 2+

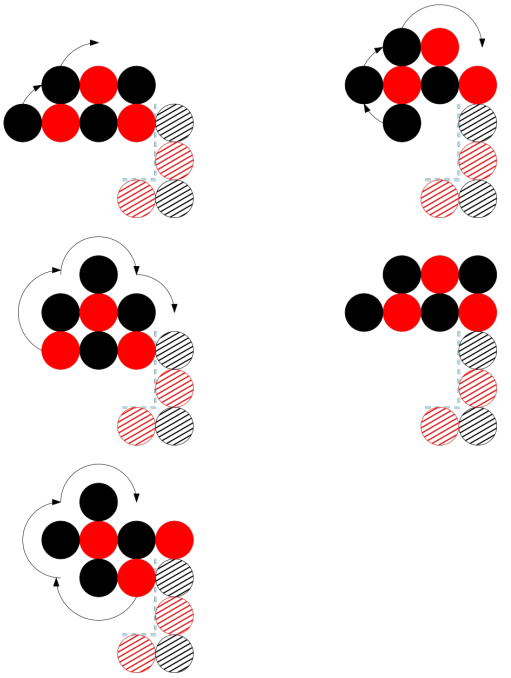


FIGURE 4.23: Climbing on top of the vertical from position 1 when the load is in the lower cell.

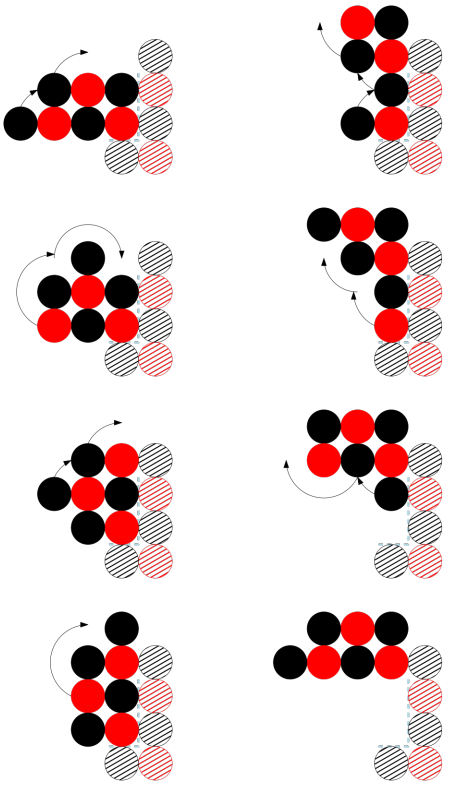


FIGURE 4.24: Climbing a vertical of height 3 from position 0 with bad colouring.

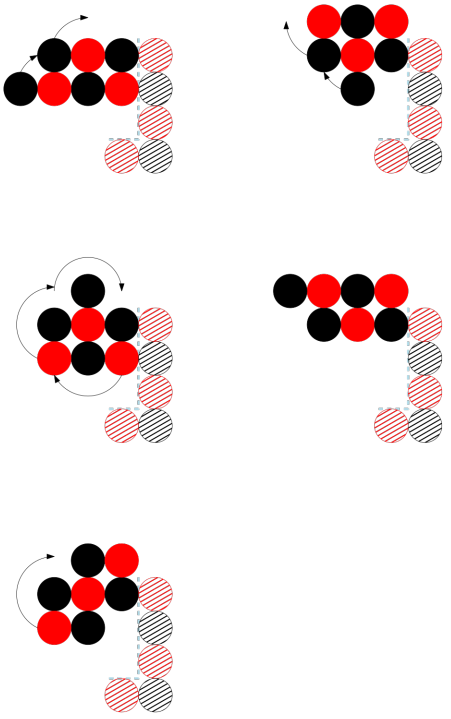


FIGURE 4.25: Climbing a vertical of height 3+ from position 2+.

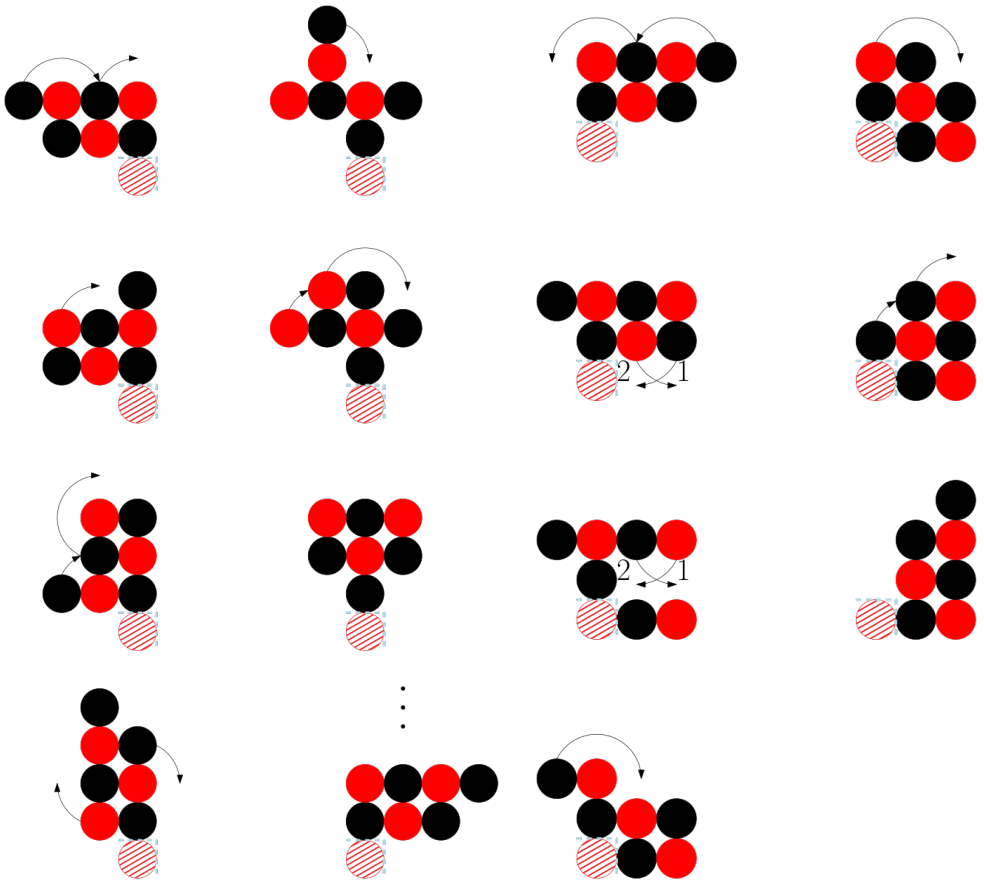


FIGURE 4.26: Movement into a new quadrant consisting of a line of length 1 when the perimeter node is not the same colour of the load. To reach the configuration after the dots, the operation is repeated in an inverted manner.

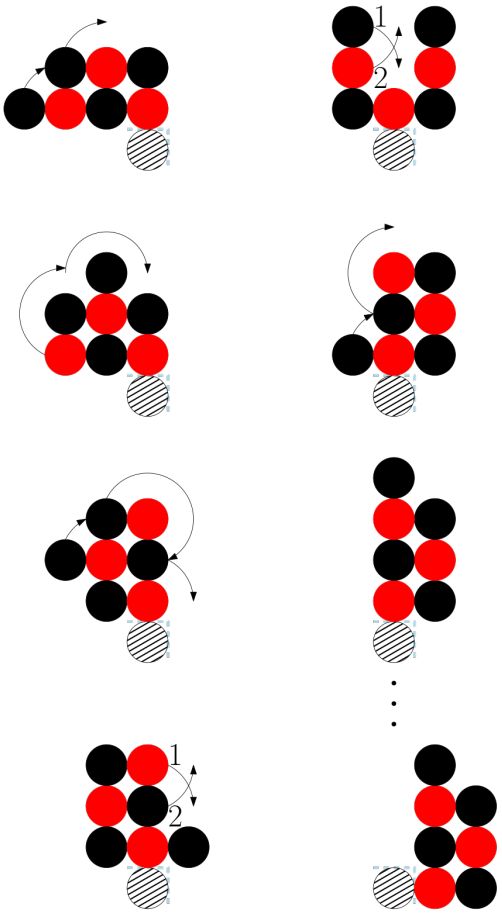


FIGURE 4.27: Movement into a new quadrant consisting of a line of length 1, with the perimeter node the same colour as the load. To reach the configuration after the dots, we follow a rotated version of the transformation in Figure 4.23.

Repository Traversal

Whenever the single-black repository is occupied, the robot may need to traverse a non-convex region when moving between S and the extended staircase. The following lemma shows that this is not an issue.

Lemma 4.15. *If the single-black repository of the extended staircase is occupied, then both the 6-robot and the 7-robot are able to traverse past it.*

Proof. We present a series of motions which for all variants of the shapes $C \in \mathcal{C}$ created by the addition of a node to the left of the vertical lead both the 6-robot and the 7-robot to the point where further movement to the topmost node of the vertical is equivalent to movement across an orthogonally convex shape. We refer to the *gap* as the set of empty cells between the single-black repository and the node below it. As before, we group some of these motions into high-level motions (i.e. moving the whole robot by moving individual nodes). When there is no node below the single-black repository, then the shape is orthogonally convex and therefore, by Theorem 4.12 and Theorem 4.14, the robot can traverse past the single-black.

Our first motion is *sliding*. If there is a 1-cell gap, then the robot can still traverse past the cell by using the special slide (Figure 4.6) for the 6-robot and the slides (Figure 4.13 and Figure 4.14) for the 7-robot, because these movements do not depend on the existence of a node in $(x_l - 2, y_d - 1)$ to provide connectivity. If there is a gap of 2 or more cells, then the robot must use special movements (Figure 4.28, Figure 4.29, Figure 4.30, Figure 4.31 and Figure 4.32) to traverse past it. The next motion is *climbing*. There are special movements (Figure 4.33) which allow the 6-robot to climb a gap of size 2, 3 and 4+. There are more movements for the 7-robot, when the gap is of size 2 (Figure 4.34), 3 (Figure 4.35) and 4+ (Figure 4.37).

We have thus shown how both types of robot can climb up and onto verticals with the additional node of any possible length. Putting everything together, the robot can use special sliding to move across a 1-cell gap, followed by special motions for larger gaps, and finally another set of special motions to climb a vertical, for all possible widths and heights. Moreover, the final position of the robot allows for the resumption of regular movement i.e. those which allow the robot to traverse an orthogonally convex shape. □

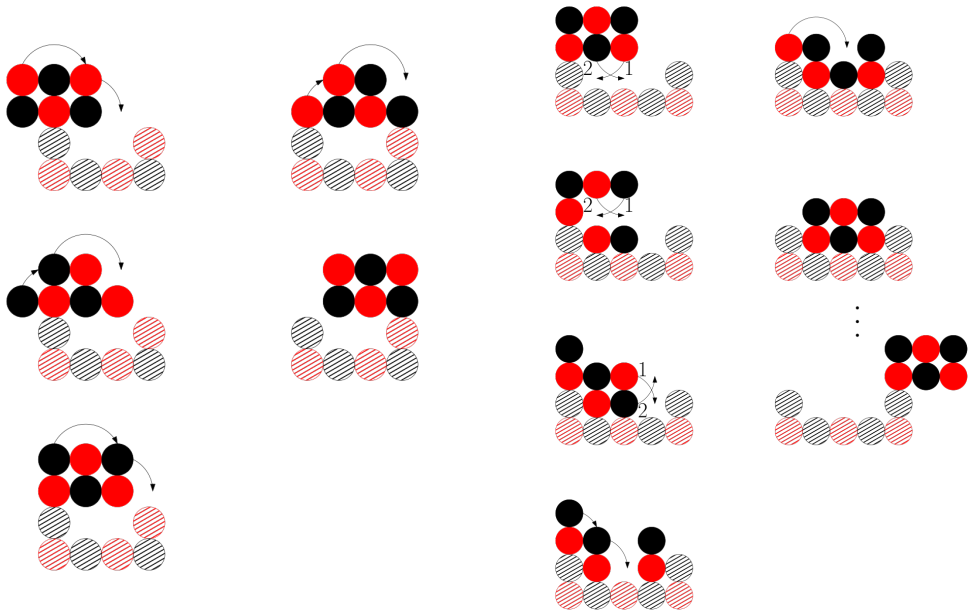


FIGURE 4.28: Sliding a 6-robot on a line with a black repository with a gap of size 2 and 3+. All movement after the final positions is equivalent to orthogonally convex movement.

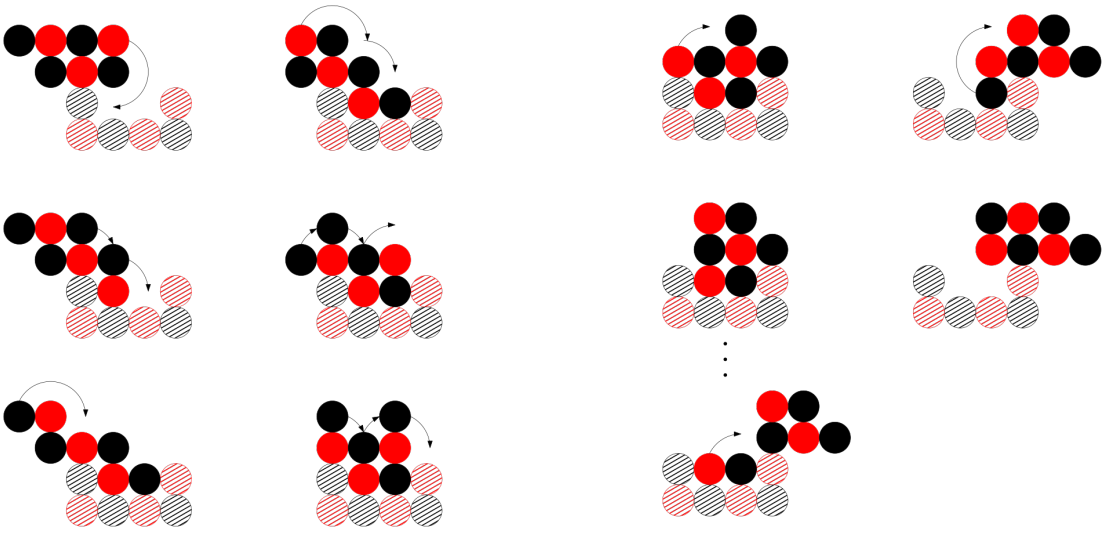


FIGURE 4.29: Sliding a 7-robot on a line with a black repository with the load in the high position, with a gap of size 2.

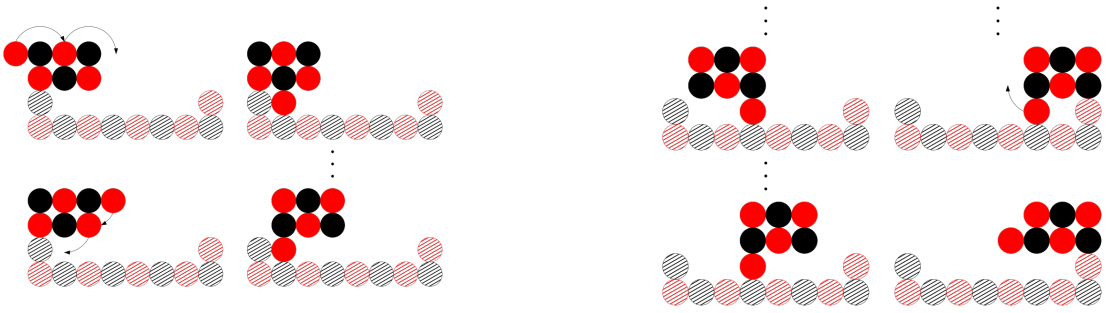


FIGURE 4.30: Sliding a 7-robot on a line with a black repository with the load in the high position, with a gap of size 3+. Note the movement after the dots can be repeated for gaps larger than 3.

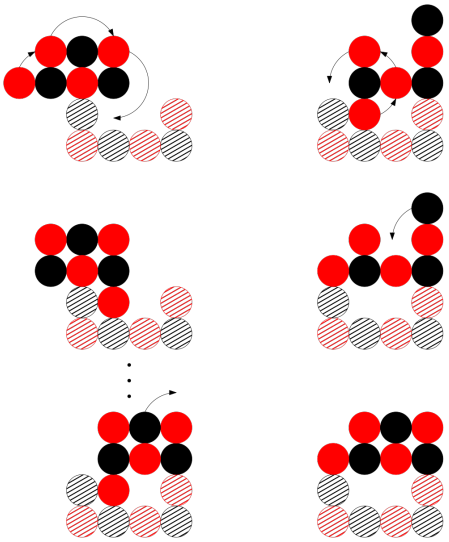


FIGURE 4.31: Sliding a 7-robot on a line with a black repository with the load in the low position, with a gap of size 2.

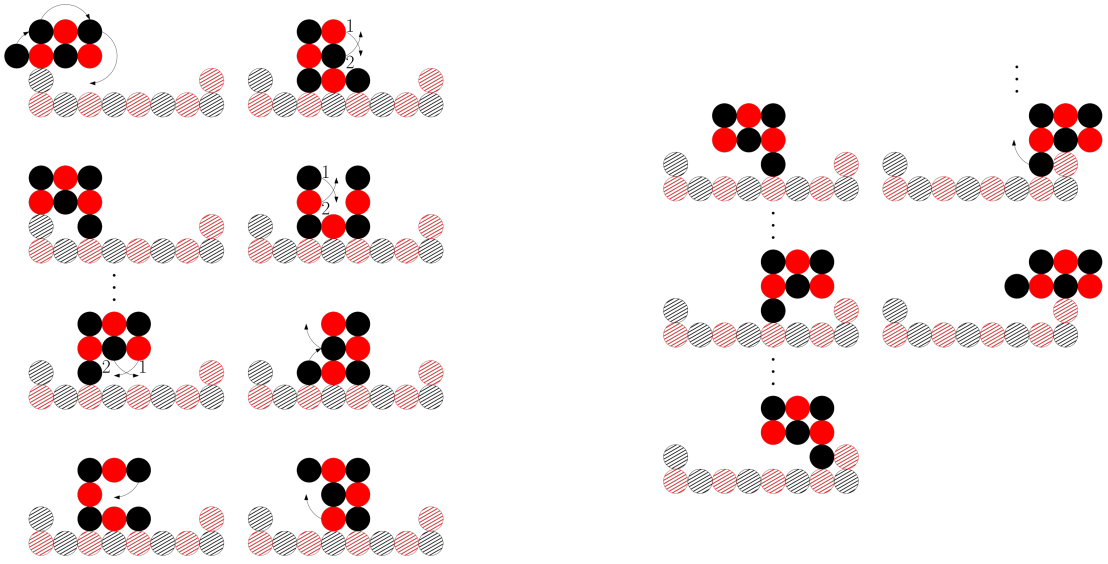


FIGURE 4.32: Sliding a 7-robot on a line with a black repository with the load in the low position, with a gap of size 3+. Note the movement after the dots can be repeated for gaps larger than 3.

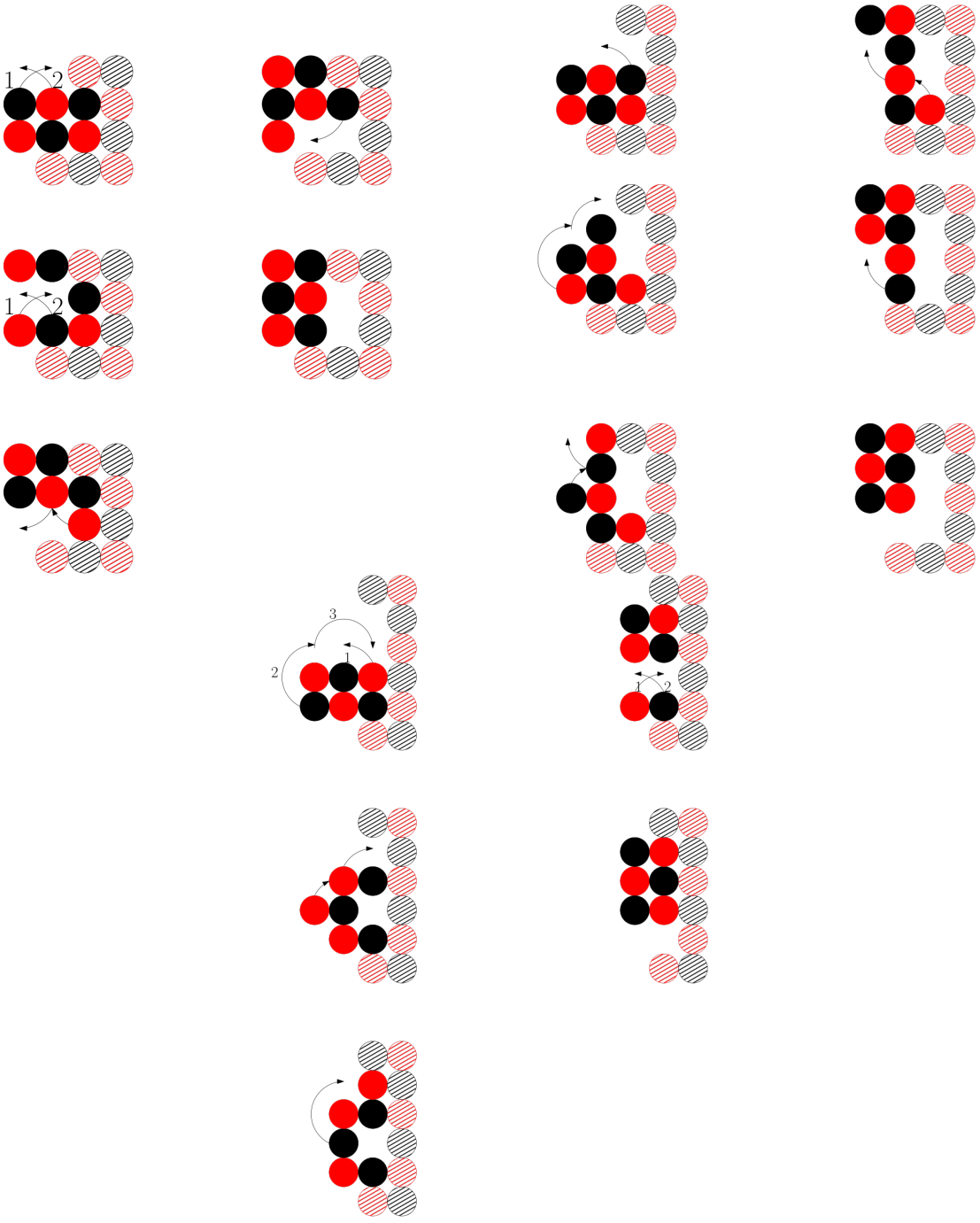


FIGURE 4.33: Climbing a 6-robot on a line with a black repository, with a gap of size 2, 3 and 4+.

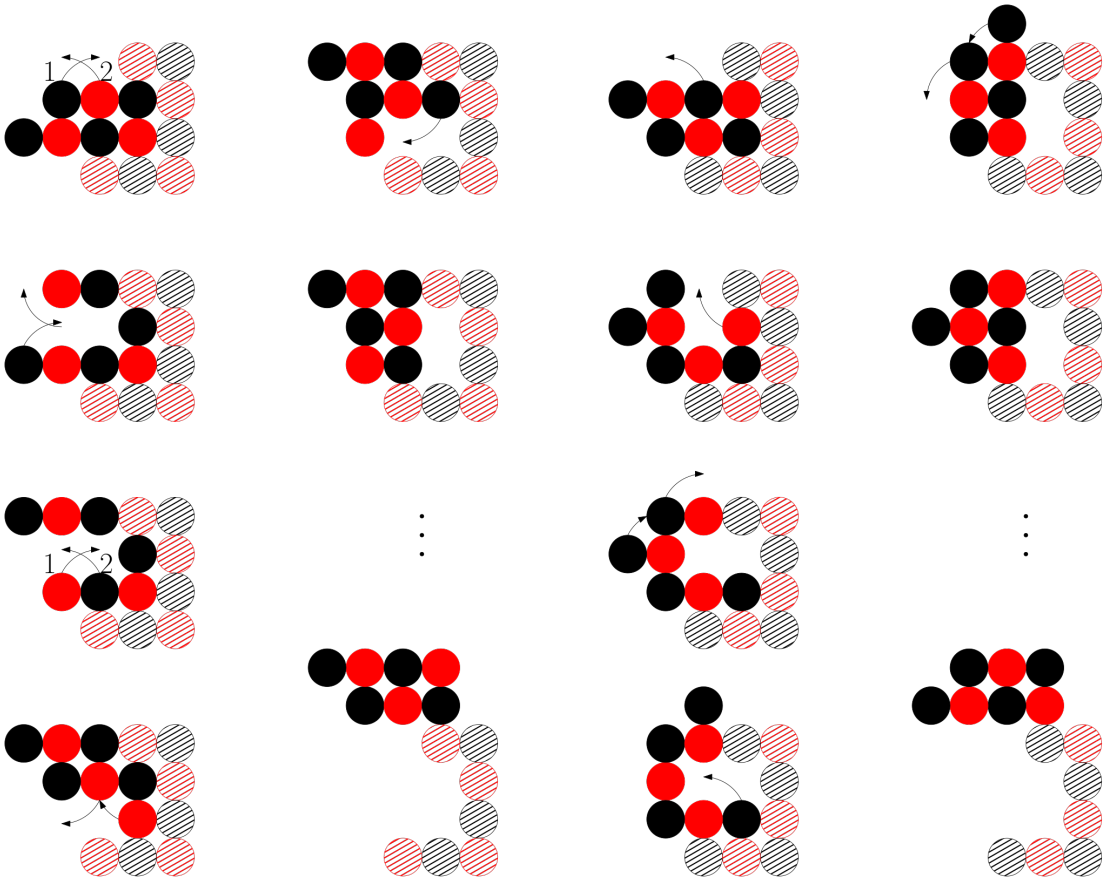


FIGURE 4.34: Climbing a 7-robot on a line with a black repository with a gap of size 2.

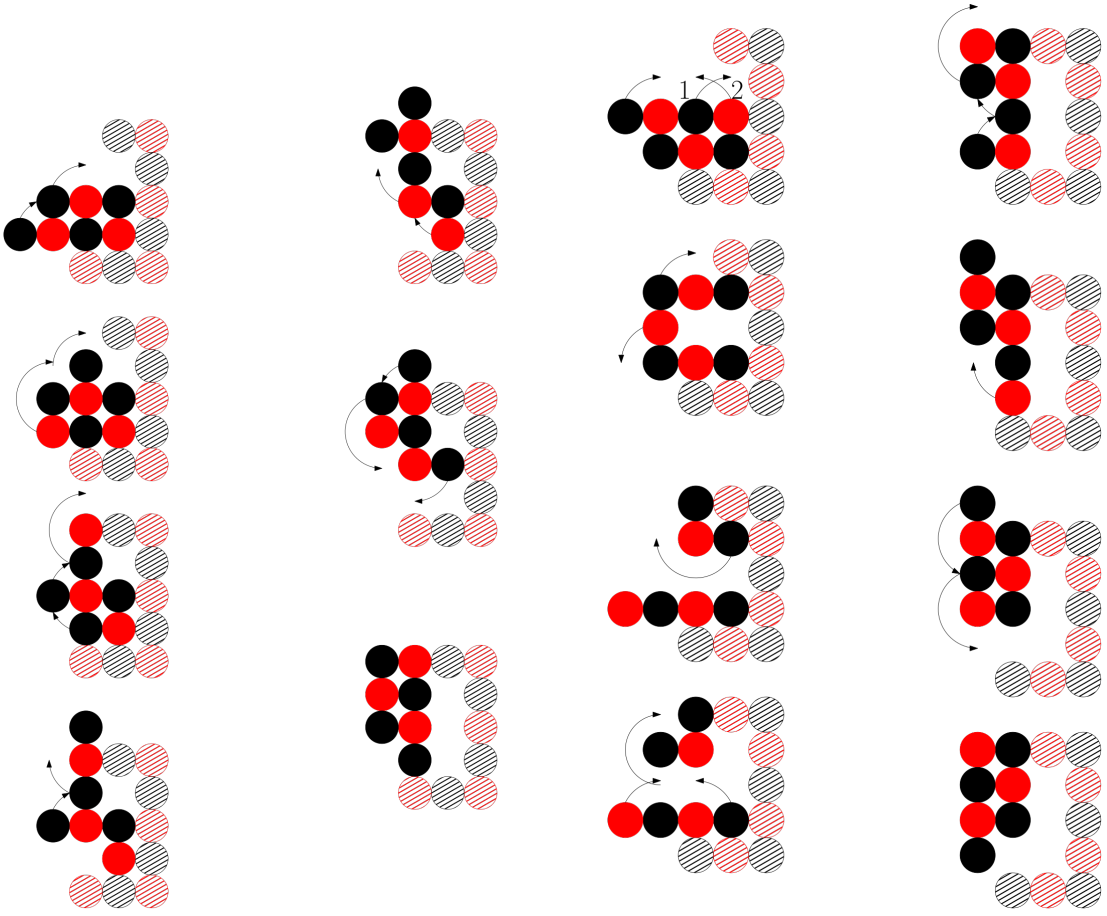


FIGURE 4.35: Climbing a 7-robot on a line with a black repository with a gap of size 3.

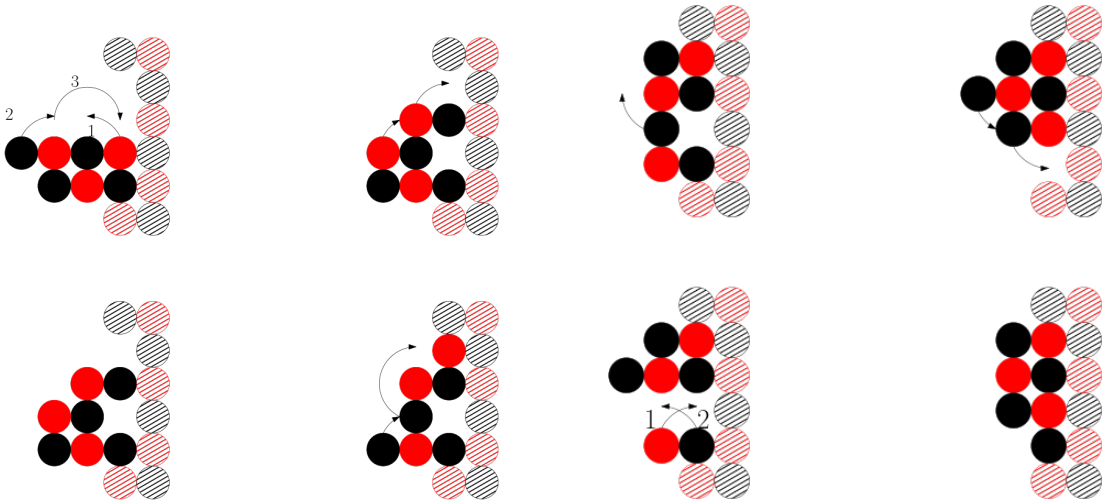


FIGURE 4.36: Climbing a 7-robot on a line with a black repository with the load in the high position, with a gap of size 4+.

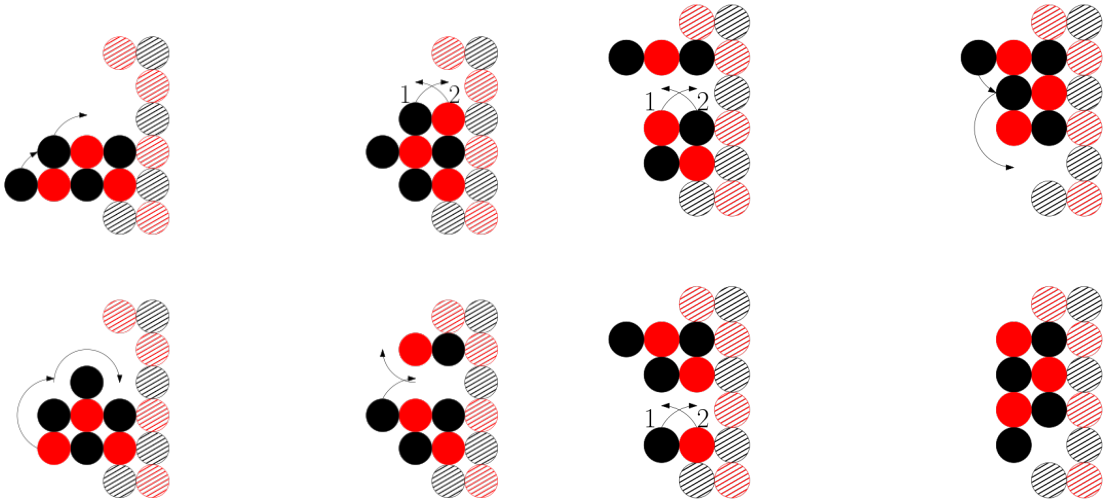


FIGURE 4.37: Climbing a 7-robot on a line with a black repository with the load in the low position, with a gap of size 4+.

4.3.2 Initialisation

Robot Generation

We now prove that we can generate a 6-robot from the orthogonally convex shape S with the help of a connected seed M of 3 nodes, which we refer to as *the 3 musketeers*.

Lemma 4.16. *Let S be a connected orthogonally convex shape. Then there is a connected shape M of 3 nodes (the 3 musketeers) and an attachment of M to the bottom-most row of S, such that $S \cup M$ can reach a configuration $S' \cup M'$ satisfying the following properties. $S' = S \setminus \{u_1, u_2, u_3\}$, where $\{u_1, u_2, u_3\}$ is the 3-prefix of a row elimination sequence σ of S starting from the bottom-most row of S. M' is a 6-robot on the perimeter of S'.*

Proof. Let R_i, $i \geq 1$, be the ith row of S counting bottom up. Assume first that $|R_1| \geq 5$ and that the elimination sequence σ can start from the rightmost node (x, y) of R_1 (the leftmost case is symmetric). If σ can continue without switching direction for at least 3 steps, then placing M as a horizontal line at $(x, y-1), (x-1, y-1), (x-2, y-1)$ gives the required 6 robot. If not, then for at least one of the two endpoints, σ can make 2 steps before switching, let that endpoint w.l.o.g. be again the rightmost node. Placing M at $(x-2, y-1), (x-3, y-1), (x-4, y-1)$ allows it to lift the two rightmost nodes, become a 5-node seed, travel to the other endpoint and lift it, thus becoming a 6-robot.

Next, let $|R_1| \in \{3, 4\}$. Then, aligning the 3 nodes of M below the rightmost 3 nodes of R_1 immediately gives the required 6-robot.

If on the other hand $|R_1| = 2$ or $|R_1| = 1$, then M can be placed so that a 5-node seed or a 4-node seed, respectively, is attached to the bottom of row R_2. If it is a 5-node seed then it can reach the rightmost/leftmost endpoint of R_2 and lift that node, thus becoming a 6-robot. If it is a 4-node seed then if $|R_2| \geq 2$ it can reach the rightmost/leftmost endpoint of R_2 and lift two nodes, possibly one node from each endpoint, thus becoming a 6-robot. The only remaining case is when a 4-node seed is attached to R_2 and $|R_2| = 1$. In that case, the configuration of the 4-node seed and the single node of R_2 can be transformed into a 5-node seed attached to the bottom of row R_3, from which the previous case can again be applied. □

Some example placements can be seen in Figure 5.1.

FIGURE 4.38: Some seed placements. The striped circles represent the orthogonally convex shape S.

Prefix Construction

The *x-gradient* and *y-gradient* of two neighbouring nodes is the difference in the x and y co-ordinates of the two nodes, respectively. A *parity rhombus* is a shape where every line is of odd length and the same colour, and the x-gradient and y-gradient of every node on the end of every line is at most 1.

To construct the extended staircase from an orthogonally convex shape S, we must first retrieve a sequence of 3 nodes u_1, u_2, u_3 from S, where u_3 is black. We assume w.l.o.g. that S is a black parity shape. We now show with the following 4 lemmas that this is possible, even in the edge case where S is a parity rhombus.

Lemma 4.17. *For any shape $S \cup M$, where S is a black parity rhombus of n nodes divided into p rows, $R_1, R_2, \ldots, R_p$ and M is a 6-robot, it is possible for M to extract two black nodes and a red node u_1, u_2, u_3 from S.*

Proof. We extract these nodes by following the procedure of Figure 4.39 and Figure 4.40. This example is for a rhombus with 5 rows but, by Theorem 4.12 and Theorem 4.14, additional rows can be navigated and so do not fundamentally change the procedure. □

An orthogonally convex shape S divided into p rows, $R_1, R_2, \ldots, R_p$ is *line-like* if the first node in R_i is above the last node in R_{i-1} for all $0 < i \leq p$.

A line l *blocks* an empty cell c in an orthogonally convex shape S if there is a node in l such that adding a node to c would cause S to lose orthogonal convexity.

Lemma 4.18. *For any shape $S \cup R$ where S is a non-red parity connected orthogonally convex shape of n nodes divided into p rows, $R_1, R_2, \ldots, R_p$ and R is a 6-robot, it is possible for R to extract a bicolour pair of nodes u, v from S, where the resulting shape $S' = S \setminus \{u, v\}$ is a connected orthogonally convex shape.*

Proof. We divide our proof into cases. In the first case, S is line-like and the rows R_1 and R_p have black nodes which can be extracted. In this case, we can extract the node and then the following node, which is necessarily of the opposite colour. If only one of R_1 and R_p has a black node which can be extracted, we can rotate S by 180° to ensure that the row containing the black does not contain the anchor node and then extract

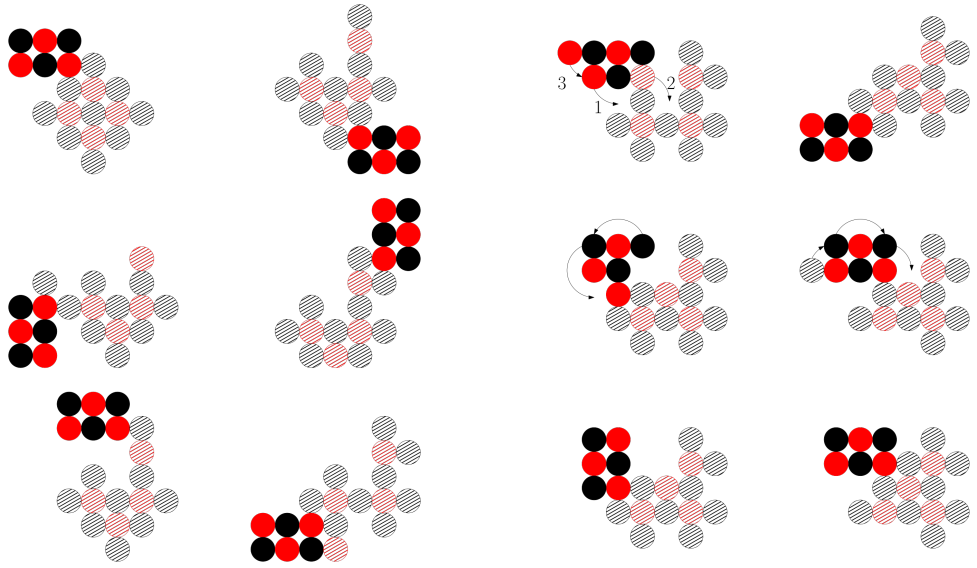


FIGURE 4.39: Converting a black parity rhombus. The figure should be read in columns, left to right (continued in Figure 4.40).

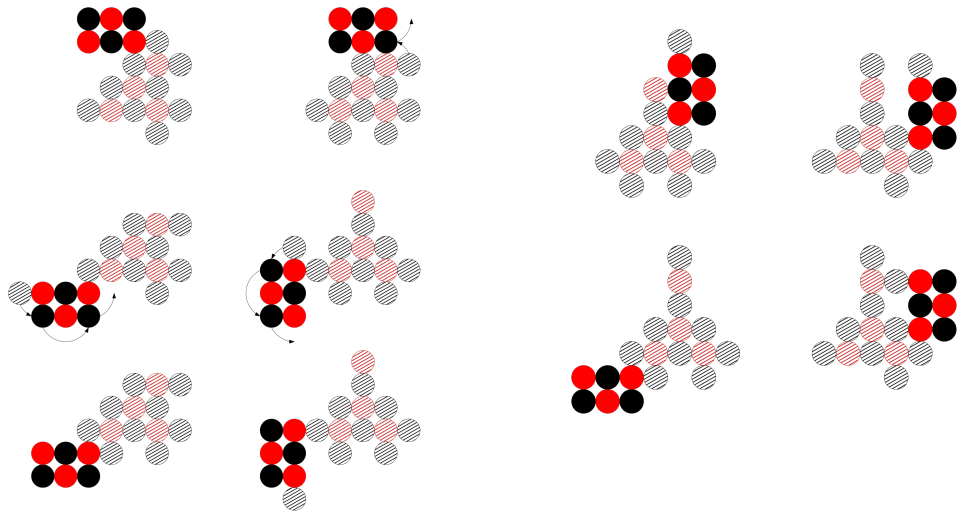


FIGURE 4.40: A continuation of Figure 4.39.

from it. If neither row has a black node which can be extracted, then the line-like shape has a red parity, which violates our assumption of a black parity shape.

If S is not line-like, then we consider the row R_p. If R_p is of length ≥ 4, then we can extract two nodes from R_p without breaking connectivity by extracting from the side furthest from the point where R_{p-1} connects to R_p. If R_p is of length 3 and R_{p-1} is of length ≥ 2, then we can extract two nodes from R_p, leaving one connected to R_{p-1}. If R_p is of length 3 and R_{p-1} is of length 1, then we can extract 2 nodes unless R_{p-1} is connected to the middle node of R_p. In this case, R_{p-2} to R_1 must be a single node and we can extract from the other end of the line starting with R_1 as the existence of a column after R_{p-1} would violate convexity. If R_p is of length 2, then we can extract R_p. If R_p and R_{p-1} are of length 1 each, then we can extract them. This leaves the cases where R_p is of length 1 and R_{p-1} is of length ≥ 1. If R_{p-1} is of length ≥ 4 then we can move R_p if necessary and extract two nodes from R_{p-1}. If R_{p-1} is of length 2 then we can rotate R_p into R_{p-1} and the resulting situation is equivalent to R_p of length 3.

If S has a single red node in R_p and R_{p-1} is of length 3, then for S to have a black parity ≥ 0, there must be some row R_i which is an odd length black line. We can move the node in R_p to either end of this line, unless there is a row R_j which blocks this, by extending further than the ends of R_i. If $R_j = R_{i+1}$ or $R_j = R_{i-1}$, then R_j cannot block R_i. If R_j is an odd length black line or even length line, we can place R_p on it. If R_j is an odd length red line, then to maintain parity, there must be another odd length black line. If there is a line R_k which neighbours R_i and has a greater length than it, then R_j cannot block R_i. Therefore, given b odd length black lines, for every line to be blocked there must be at least $r = b + 1$ odd length red lines, one long line to block the nodes and b lines to connect them together into a shape. Including R_p, such a shape would have a red parity of at least 2, and is therefore impossible via assumption. We can therefore move R_p, creating a new situation where extraction of two nodes from R_{p-1} is possible.

If S has a single black node in R_p and R_{p-1} is of length 3, then if R_{p-2} is of length 2 we can move the node in R_p to it and extract from R_{p-1}. If R_{p-2} is also of a length ≥ 3 then unless S is a black parity rhombus it is possible for the 6-robot to extract the node in R_p and move away from R_{p-1}. After that, the red node from R_{p-1} can be rotated. The robot can then store the black node it is carrying on the red node by moving around the perimeter of the shape, which is still orthogonally convex. Finally, the 6-robot can extract both the black node and the red node. If S is a black parity rhombus then by Lemma 4.17 we can extract two nodes from it using special movements. □

Lemma 4.19. *For any shape $S \cup M$, where S is a non-red parity connected orthogonally convex shape of n nodes divided into p rows, $R_1, R_2, \ldots, R_p$ and M is a 6-robot, it is*

possible for M to extract a black node u from S, where the resulting shape $S' = S \setminus \{u\}$ is a connected orthogonally convex shape.

Proof. We consider two cases for when the black nodes are the majority, and when they are exactly $n/2$.

In the first case, since S is majority black by assumption, there must be at least one row R_i which is a single black node or a line of odd length which begins and ends with a black node. We can extract a black node from this line, unless such an extraction will violate orthogonal convexity. This occurs if the neighbouring lines R_{i-1} and R_{i+1} are of the same length. For S to have black parity, this implies the existence of another odd length line from which a black node can be extracted. More generally, given x odd length lines which end in reds, by the pigeonhole principle there must be $x+1$ odd length lines ending in blacks, implying at least one can be extracted from without violating orthogonal convexity. Moreover, the removal cannot break connectivity, because this would imply R_{i-1} and R_{i+1} are single red nodes on either end of R_i, which also implies the existence of another odd length line to maintain the ratio of black to red nodes.

In the second case, we try to extract a black node from the bottom row R_p. If this is not possible, it implies either that R_p is an odd length line with red parity or R_p is an even length line and R_{p-1} is a single red node connected to the black node at the end of R_p. In either case, the existence of a row which is of red parity implies the existence of an odd length line of black parity to maintain the overall parity of the shape. A similar pigeonhole principle argument to the first case follows, both for orthogonal convexity and connectivity.

There is a special case where the black node to be extracted happens to be the anchor node. In this scenario, we simply rotate the shape 180°, giving us an equivalent scenario where the black node is in R_p and thus guaranteed to be accessible.

□

Lemma 4.20. *For any shape $S \cup M$, where S is a non-red parity connected orthogonally convex shape of n nodes divided into p rows, $R_1, R_2, \ldots, R_p$ and M is a 6-robot, it is possible for M to extract a sequence of nodes (u_1, u_2, u_3) from S, where u_1, u_2 is a bicolour pair, u_3 is black, and $S \setminus \{u_1, u_2, u_3\}$ is a connected orthogonally convex shape.*

Proof. By Lemma 4.18, we can extract two nodes from a S. The 6-robot places these nodes on the anchor node. By Lemma 4.19 we can then extract a black node from S. The 6-robot places this node as well. □

4.3.3 Transformations Between Shapes

In this section, we show that, given our previous results, we are now in the position to convert an orthogonally convex shape into another such shape. We begin with the

conversion of an extended staircase into a diagonal line-with-leaves, then the orthogonally convex shape to the diagonal line-with-leaves, and then our main result follows by reversibility.

Transforming S to Extended Staircase

Lemma 4.21. *Let S be a connected orthogonally convex shape with n nodes divided into p rows $R_1, R_2, \ldots, R_p$. Given a row elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$ of S, an extended staircase generation sequence $\sigma' = (u'_1, u'_2, \ldots, u'_n)$ which is colour-order preserving w.r.t. σ, and a 6-robot placed on the external surface of S, for all $1 \leq i < n$ the 6-robot is capable of picking up the node u_i, moving as a 7-robot to the empty cell u'_i and placing it, and then returning as a 6-robot to u_{i+1}.*

Proof. We follow the procedure of Algorithm 4. By Theorem 4.12 and Theorem 4.14, the 6-robot R and 7-robot $R \cup u_i$ can climb and slide around the external surface of S. We use this to move to each u_i, extract it, move to the cell for u'_i and then place a node of the same colour as u_i in it, substituting u_i for a node in R as necessary to create a new 6-robot. By Lemma 4.8, so long as we approach T_i from R_p, we can climb onto and off T_i to place the nodes using the same movements as the previous theorems. By Lemma 4.15, placing a black node in the repository cell does not inhibit movement. □

Transforming Extended Staircase to Diagonal Line-with-Leaves

Lemma 4.22. *Let $W \cup T \cup R$ be the union of the Stairs of an extended staircase W, $T \subseteq \{BRep \cup RRep\}$ from the extended staircase and a 6-node robot R on the cell perimeter of $S \cup T$. Given a shape elimination sequence $\sigma = (u_1, u_2, \ldots, u_n)$ of T, a diagonal line-with-leaves generation sequence σ' which is colour-order preserving w.r.t. σ and a 6-robot placed on the external surface of S, for all $1 \leq i \leq n$ the 6-robot is capable of picking up the node u_i, moving as a 7-robot to u'_i and placing it, and then returning as a 6-robot to u_{i+1}.*

Proof. We follow the procedure of Algorithm 5. By Theorem 4.12 and Theorem 4.14, the 6-robot R and 7-robot $R \cup u_i$ can climb and slide around the external surface of $S \cup T$. We use this to move to each u_i, extract it, move to the cell for u'_i and then place a node of the same colour as u_i in it, substituting u_i for a node in R as necessary to create a new 6-robot. Since the placement of u'_i is extending *Stairs*, the resulting shape is always orthogonally convex for all $1 \leq i \leq n$. □

Transforming S to Diagonal Line-with-Leaves

Lemma 4.23. *Let S be a connected orthogonally convex shape. Then there is a connected shape M of 3 nodes (the 3 musketeers) and an attachment of M to the bottom-most row of S, such that $S \cup M$ can reach the configuration D, where D is a diagonal line-with-leaves which is colour-consistent with S.*

Proof. We follow the procedure of Algorithm 3. By Lemma 4.16 we can form a 6-robot from $S \cup M$. By Lemma 4.21, we can build an extended staircase from the resulting shape. By Lemma 4.22, we can then build a diagonal line-with-leaves. Finally, by reversibility, we can place R such that the removal of 3 nodes leaves a larger diagonal line-with-leaves D which is colour-consistent with S. □

4.3.4 Time Analysis and Wrapping Up

Lemma 4.24. *For any connected orthogonally convex shape S of n nodes, there exists a corresponding diagonal line-with-leaves T such that our strategy transforms S to T in $O(n^2)$ time steps.*

Proof. To construct T, we transfer nodes using the robot to the anchor node. In the worst case, S is a staircase, and the robot must move nodes from one end to the other. It must therefore make $O(cn^2)$ moves, where c is the constant number of rotations needed for the robot to move one step. When the extended staircase has been constructed, it is converted into a diagonal line-with-leaves. In the worst case every column in the staircase has 4 nodes, and the robot must extend *Stairs* until one repository has a single node. Therefore, the robot must make $O(2cn^2)$ moves to travel on both sides of *Stairs*. Combining the worst cases of both procedures therefore takes $O(3cn^2) = O(n^2)$ time steps. □

Proposition 4.25. *For any two connected orthogonally convex shapes S and T which are colour-consistent, Algorithm 3 generates the diagonal line-with-leaves D and G such that $D = G$.*

Theorem 4.26. *Let S and S' be connected colour-consistent orthogonally convex shapes. Then there is a connected shape M of 3 nodes (the 3 musketeers) and an attachment of M to the bottom-most row of S, such that $S \cup M$ can reach the configuration S' in $O(n^2)$ time steps.*

Proof. By Lemma 4.23, we can convert S into a diagonal line-with-leaves T. By reversibility, we can convert T into S'. By Lemma 4.24, this procedure takes $O(n^2)$ time steps. □

Chapter 5

Conclusions and Future Work

There are some open problems which follow from the findings of our work. The most obvious is expanding the class of shapes which can be constructed to achieve universal transformation. An example of a challenging instance is the "double spiral", which is a line forming two connected spirals, where each of the two endpoints of the line is concealed at the centre of a spiral and the removal of any other node would break connectivity. In this case, preserving connectivity after the removal of a node requires the robot to get to the centre of a spiral, which may not be possible without a special procedure to "dig" into it without breaking connectivity. Finally, successfully switching to a decentralised model of transformations will greatly expand the utility of the results, especially because most programmable matter systems which model real-world applications implement programs in this way. This in turn could lead to real-world applications for the efficient transformation of programmable matter systems.

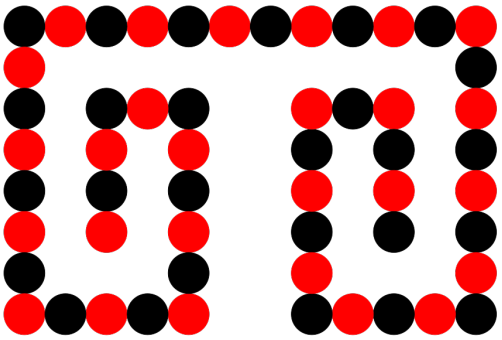


FIGURE 5.1: An example of the double spiral shape.

Bibliography

[1] Zykov V, William P, N. Lassabe, and Hod Lipson. Molecubes extended: Diversifying capabilities of open-source modular robotics. In *IEEE Robotics and Automation Society Self-Reconfigurable Robotics Workshop*, 01 2008.

[2] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, August 2014. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1254295.

[3] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *2010 IEEE International Conference on Robotics and Automation*, pages 2485–2492, May 2010. doi: 10.1109/ROBOT.2010.5509817. ISSN: 1050-4729.

[4] Ara N. Knaian, Kenneth C. Cheung, Maxim B. Lobovsky, Asa J. Oines, Peter Schmidt-Neilsen, and Neil A. Gershenfeld. The Milli-Motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1447–1453, October 2012. doi: 10.1109/IROS.2012.6385904. ISSN: 2153-0866.

[5] Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 140–148, Montreal QC, Canada, May 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.

[6] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. 3D Coating Self-Assembly for Modular Robotic Scaffolds. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11688–11695, October 2020. doi: 10.1109/IROS45743.2020.9341324. ISSN: 2153-0866.

[7] Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected Coordinated Motion Planning with Bounded Stretch.

Technical Report arXiv:2109.12381, arXiv, September 2021. URL `http://arxiv.org/abs/2109.12381`. arXiv:2109.12381 [cs] type: article.

[8] Benoît Piranda and Julien Bourgeois. Datom: A Deformable Modular Robot for Building Self-reconfigurable Programmable Matter. In Fumitoshi Matsuno, Shun-ichi Azuma, and Masahito Yamamoto, editors, *Distributed Autonomous Robotic Systems*, Springer Proceedings in Advanced Robotics, pages 70–81, Cham, 2022. Springer International Publishing. ISBN 9783030927905. doi: 10.1007/978-3-030-92790-5_6.

[9] Benoît Piranda and Julien Bourgeois. Datom: A Deformable Modular Robot for Building Self-reconfigurable Programmable Matter. In Fumitoshi Matsuno, Shun-ichi Azuma, and Masahito Yamamoto, editors, *Distributed Autonomous Robotic Systems*, Springer Proceedings in Advanced Robotics, pages 70–81, Cham, 2022. Springer International Publishing. ISBN 9783030927905. doi: 10.1007/978-3-030-92790-5_6.

[10] Gianlorenzo D'Angelo, Mattia D'Emidio, Shantanu Das, Alfredo Navarra, and Giuseppe Prencipe. Asynchronous Silent Programmable Matter Achieves Leader Election and Compaction. *IEEE Access*, 8:207619–207634, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3038174.

[11] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 353–354, Berkeley, California, USA, January 2013. Association for Computing Machinery. ISBN 9781450318594. doi: 10.1145/2422436.2422476.

[12] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, SPAA '14, pages 220–222, Prague, Czech Republic, June 2014. Association for Computing Machinery. ISBN 9781450328210. doi: 10.1145/2612669.2612712.

[13] Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, March 2018. ISSN 1572-9796. doi: 10.1007/s11047-017-9658-6.

[14] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An Algorithmic Framework for Shape Formation Problems

in Self-Organizing Particle Systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, NANOCOM' 15, pages 1–2, Boston, MA, USA, September 2015. Association for Computing Machinery. ISBN 9781450336741. doi: 10.1145/2800795.2800829.

[15] Zahra Derakhshandeh, Robert Gmyr, Andrea W. Richa, Christian Scheideler, and Thim Strothmann. Universal Shape Formation for Programmable Matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 289–299, Pacific Grove, California, USA, July 2016. Association for Computing Machinery. ISBN 9781450342100. doi: 10.1145/2935764.2935784.

[16] Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating Amoebots via Reconfigurable Circuits. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, 29(4):317–343, April 2022. ISSN 1557-8666. doi: 10.1089/cmb.2021.0363.

[17] Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. Technical Report arXiv:2105.02420, arXiv, May 2021. URL `http://arxiv.org/abs/2105.02420`. arXiv:2105.02420 [cs] type: article.

[18] Irina Kostitsyna, Christian Scheideler, and Daniel Warner. Brief Announcement: Fault-Tolerant Shape Formation in the Amoebot Model. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*, volume 221 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:3, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 9783959772242. doi: 10.4230/LIPIcs.SAND.2022.23. URL `https://drops.dagstuhl.de/opus/volltexte/2022/15965`.

[19] Greg Aloupis, Sebastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristan, and Stefanie Wuhrer. Reconfiguration of 3D Crystalline Robots Using O(log n) Parallel Moves, August 2009. URL `http://arxiv.org/abs/0908.2440`. arXiv:0908.2440 [cs].

[20] Othon Michail, George Skretas, and Paul G. Spirakis. Distributed computation and reconfiguration in actively dynamic networks. *Distributed Computing*, 35(2): 185–206, April 2022. ISSN 1432-0452. doi: 10.1007/s00446-021-00415-5. URL `https://doi.org/10.1007/s00446-021-00415-5`.

[21] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications.

ACM SIGCOMM Computer Communication Review, 31(4):149–160, August 2001. ISSN 0146-4833. doi: 10.1145/964723.383071. URL https://doi.org/10.1145/964723.383071.

[22] Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In *26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 262–276, 2019.

[23] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and Inference Problems for Temporal Networks. *Journal of Computer and System Sciences*, 64(4):820–842, June 2002. ISSN 0022-0000. doi: 10.1006/jcss.2002.1829. URL https://www.sciencedirect.com/science/article/pii/S0022000002918295.

[24] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Solving the Robots Gathering Problem. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 1181–1196, Berlin, Heidelberg, 2003. Springer. ISBN 9783540450610. doi: 10.1007/3-540-45061-0_90.

[25] Evangelos Kranakis, Danny Krizanc, and Euripides Markou. Multiple Agent Rendezvous in a Ring. In Evangelos Kranakis, Danny Krizanc, and Euripides Markou, editors, *The Mobile Agent Rendezvous Problem in the Ring*, Synthesis Lectures on Distributed Computing Theory, pages 27–33. Springer International Publishing, Cham, 2010. ISBN 9783031019999. doi: 10.1007/978-3-031-01999-9_3. URL https://doi.org/10.1007/978-3-031-01999-9_3.

[26] Masahiro Shibata, Toshiya Mega, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Uniform Deployment of Mobile Agents in Asynchronous Rings. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 415–424, New York, NY, USA, July 2016. Association for Computing Machinery. ISBN 9781450339643. doi: 10.1145/2933057.2933093. URL https://doi.org/10.1145/2933057.2933093.

[27] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Springer International Publishing, Cham, 2012. ISBN 9783031008801 9783031020087. doi: 10.1007/978-3-031-02008-7. URL https://link.springer.com/10.1007/978-3-031-02008-7.

[28] Ichiro Suzuki and Masafumi Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal on Computing*, 28(4):1347–1363, January 1999. ISSN 0097-5397. doi: 10.1137/S009753979628292X. URL https://epubs.siam.org/doi/10.1137/S009753979628292X.

[29] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, April 2015. ISSN 1432-0452. doi: 10.1007/s00446-014-0220-9. URL `https://doi.org/10.1007/s00446-014-0220-9`.

[30] Alejandro Cornejo, Fabian Kuhn, Ruy Ley-Wild, and Nancy Lynch. Keeping Mobile Robot Swarms Connected. In Idit Keidar, editor, *Distributed Computing*, Lecture Notes in Computer Science, pages 496–511, Berlin, Heidelberg, 2009. Springer. ISBN 9783642043550. doi: 10.1007/978-3-642-04355-0_50.

[31] Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, June 2019. ISSN 0022-0000. doi: 10.1016/j.jcss.2018.12.001.

[32] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

[33] Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 459–468, Portland, Oregon, USA, May 2000. Association for Computing Machinery. ISBN 9781581131840. doi: 10.1145/335305.335358.

[34] Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, June 2014. ISSN 1572-9796. doi: 10.1007/s11047-013-9379-4. URL `https://doi.org/10.1007/s11047-013-9379-4`.

[35] David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, December 2012. ISSN 0001-0782. doi: 10.1145/2380656.2380675.

[36] Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, March 2006. ISSN 1476-4687. doi: 10.1038/nature04586.

[37] Damien Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, 373(2046):20140214, July 2015. ISSN 1471-2962. doi: 10.1098/rsta.2014.0214.

[38] Ke Li, Kyle Thomas, Louis F. Rossi, and Chien-Chung Shen. Slime Mold Inspired Protocol for Wireless Sensor Networks. In *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 319–328, October 2008. doi: 10.1109/SASO.2008.58. ISSN: 1949-3681.

[39] Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. *Journal of Theoretical Biology*, 309:121–133, September 2012. ISSN 0022-5193. doi: 10.1016/j.jtbi.2012.06.017. URL `https://www.sciencedirect.com/science/article/pii/S0022519312003049`.

[40] Devleena Samanta, Wenjie Zhou, Sasha B. Ebrahimi, Sarah Hurst Petrosko, and Chad A. Mirkin. Programmable Matter: The Nanoparticle Atom and DNA Bond. *Advanced Materials*, 34(12):2107875, March 2022. ISSN 0935-9648, 1521-4095. doi: 10.1002/adma.202107875. URL `https://onlinelibrary.wiley.com/doi/10.1002/adma.202107875`.

[41] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006. ISSN 1432-0452. doi: 10.1007/s00446-005-0138-3.

[42] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, July 2007.

[43] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing (STOC)*, pages 513–522, 2010.

[44] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *16th International Conference on Principles of Distributed Systems (OPODIS)*, pages 269–283, 2012.

[45] David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Verification and computation in restricted Tile Automata. *Natural Computing*, November 2021. ISSN 1572-9796. doi: 10.1007/s11047-021-09875-x. URL `https://doi.org/10.1007/s11047-021-09875-x`.

[46] Yuval Emek and Jara Uitto. Dynamic networks of finite state machines. *Theoretical Computer Science*, 810:58–71, March 2020. ISSN 0304-3975. doi: 10.1016/j.tcs.2017.05.025. URL `https://www.sciencedirect.com/science/article/pii/S0304397517304668`.

[47] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, December 2008. ISSN 1572-9796. doi: 10.1007/s11047-008-9067-y.

[48] David Doty. Timing in chemical reaction networks. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 772–784. Society for Industrial and Applied Mathematics, December 2013. ISBN 9781611973389. doi: 10.1137/1.9781611973402.57.

[49] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity.*, volume 1. Basic Books, 1996.

[50] Nooshin Nokhanji, Paola Flocchini, and Nicola Santoro. Fully Dynamic Line Maintenance by a Simple Robot. In *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*, pages 108–112, February 2022. doi: 10.1109/ICARA55094.2022.9738561. ISSN: 2767-7745.

[51] Othon Michail and Paul G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016. ISSN 0178-2770. doi: http://dx.doi.org/10.1007/s00446-015-0257-4. URL `https://link.springer.com/article/10.1007/s00446-015-0257-4`.

[52] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theoretical Computer Science*, 412(22):2434–2450, May 2011.

[53] Othon Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. *Distributed Computing*, 31(5):343–365, 2018. ISSN 0178-2770. doi: http://dx.doi.org/10.1007/s00446-017-0309-z. URL `http://link.springer.com/article/10.1007/s00446-017-0309-z`.

[54] Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 137:1–137:15, 2017.

[55] Jurek Czyzowicz, Dariusz Dereniowski, and Andrzej Pelc. Building a Nest by an Automaton. *Algorithmica*, 83(1):144–176, 2021. ISSN 0178-4617. doi: 10.1007/s00453-020-00752-0.

[56] Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov Chain Algorithm for Compression in Self-Organizing Particle Systems. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 279–288, New York, NY, USA, July 2016. Association for Computing Machinery. ISBN 9781450339643. doi: 10.1145/2933057.2933107. URL `https://doi.org/10.1145/2933057.2933107`.

[57] Joshua J. Daymude, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Improved Leader Election for Self-organizing Programmable

Matter. In Antonio Fernández Anta, Tomasz Jurdzinski, Miguel A. Mosteiro, and Yanyong Zhang, editors, *Algorithms for Sensor Systems*, Lecture Notes in Computer Science, pages 127–140, Cham, 2017. Springer International Publishing. ISBN 9783319727516. doi: 10.1007/978-3-319-72751-6_10.

[58] Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by Programmable Particles. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, Lecture Notes in Computer Science, pages 615–681. Springer International Publishing, Cham, 2019. ISBN 9783030110727. doi: 10.1007/978-3-030-11072-7_22. URL https://doi.org/10.1007/978-3-030-11072-7_22.

[59] Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1): 81–96, March 2018. ISSN 1572-9796. doi: 10.1007/s11047-017-9658-6. URL https://doi.org/10.1007/s11047-017-9658-6.

[60] Joshua J. Daymude, Andréa W. Richa, and Jamison W. Weber. Bio-Inspired Energy Distribution for Programmable Matter. In *International Conference on Distributed Computing and Networking 2021*, ICDCN '21, pages 86–95, New York, NY, USA, January 2021. Association for Computing Machinery. ISBN 9781450389334. doi: 10.1145/3427796.3427835. URL https://doi.org/10.1145/3427796.3427835.

[61] Sándor P. Fekete, Robert Gmyr, Sabrina Hugo, Phillip Keldenich, Christian Scheffer, and Arne Schmidt. CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata. *Algorithmica*, 83(1):387–412, January 2021. ISSN 1432-0541. doi: 10.1007/s00453-020-00761-z. URL https://doi.org/10.1007/s00453-020-00761-z.

[62] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007. ISSN 1432-0452. doi: 10.1007/s00446-007-0040-2. URL https://doi.org/10.1007/s00446-007-0040-2.

[63] Dana Angluin, James Aspnes, and David Eisenstat. Fast Computation by Population Protocols with a Leader. In Shlomi Dolev, editor, *Distributed Computing*, Lecture Notes in Computer Science, pages 61–75, Berlin, Heidelberg, 2006. Springer. ISBN 9783540446279. doi: 10.1007/11864219_5.

[64] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing Population Protocols. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *Principles of Distributed Systems*, Lecture Notes in Computer Science, pages 103–117, Berlin, Heidelberg, 2006. Springer. ISBN 9783540363224. doi: 10.1007/11795490_10.

[65] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When Birds Die: Making Population Protocols Fault-Tolerant. In Phillip B. Gibbons, Tarek Abdelzaher, James Aspnes, and Ramesh Rao, editors, *Distributed Computing in Sensor Systems*, Lecture Notes in Computer Science, pages 51–66, Berlin, Heidelberg, 2006. Springer. ISBN 9783540352280. doi: 10.1007/11776178_4.

[66] Adrian Dumitrescu and János Pach. Pushing Squares Around. *Graphs and Combinatorics*, 22(1):37–50, April 2006. ISSN 1435-5914. doi: 10.1007/s00373-005-0640-1. URL `https://doi.org/10.1007/s00373-005-0640-1`.

[67] M. A. McEvoy and N. Correll. Materials that couple sensing, actuation, computation, and communication. *Science*, 347(6228):1261689, March 2015. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1261689. URL `https://www.science.org/doi/10.1126/science.1261689`.

[68] Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72, January 2018. ISSN 0001-0782. doi: 10.1145/3156693. URL `https://doi.org/10.1145/3156693`.

[69] J.E. Walter, J.L. Welch, and N.M. Amato. Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. *IEEE Transactions on Robotics and Automation*, 18(6):945–956, December 2002. ISSN 2374-958X. doi: 10.1109/TRA.2002.805648.

[70] Yukiko Yamauchi, Taichi Uehara, Shuji Kijima, and Masafumi Yamashita. Plane Formation by Synchronous Mobile Robots in the Three-Dimensional Euclidean Space. *Journal of the ACM*, 64(3):16:1–16:43, June 2017. ISSN 0004-5411. doi: 10.1145/3060272. URL `https://doi.org/10.1145/3060272`.

[71] Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science*, 830-831:43–59, August 2020. ISSN 0304-3975. doi: 10.1016/j.tcs.2020.04.026.

[72] Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van

Renssen, and Vera Sacristán. Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The O(1) Musketeers. *Algorithmica*, 83(5):1316–1351, May 2021. ISSN 1432-0541. doi: 10.1007/s00453-020-00784-6.

[73] Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for Fast Locomotion of Metamorphic Robotic Systems. *The International Journal of Robotics Research*, 23(6):583–593, June 2004. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364904039652. URL `http://journals.sagepub.com/doi/10.1177/0278364904039652`.

[74] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20(3):409–418, June 2004. ISSN 2374-958X. doi: 10.1109/TRA.2004.824936.

[75] Abdullah Almethen, Othon Michail, and Igor Potapov. Distributed Transformations of Hamiltonian Shapes Based on Line Moves. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Algorithms for Sensor Systems*, Lecture Notes in Computer Science, pages 1–16, Cham, 2021. Springer International Publishing. ISBN 9783030892401. doi: 10.1007/978-3-030-89240-1_1.

[76] Abdullah Almethen, Othon Michail, and Igor Potapov. On efficient connectivity-preserving transformations in a grid. *Theoretical Computer Science*, 898:132–148, January 2022. ISSN 0304-3975. doi: 10.1016/j.tcs.2021.11.004. URL `https://www.sciencedirect.com/science/article/pii/S0304397521006514`.

[77] Irina Kostitsyna, Tom Peters, and Bettina Speckmann. Fast reconfiguration for programmable matter. In *The 38th European Workshop on Computational Geometry*, pages 365–371, 2022.

[78] Othon Michail, George Skretas, and Paul G. Spirakis. On the Transformation Capability of Feasible Mechanisms for Programmable Matter. Technical Report arXiv:1703.04381, arXiv, March 2017. URL `http://arxiv.org/abs/1703.04381`. arXiv:1703.04381 [cs] type: article.

[79] Yukiko Yamauchi. Symmetry of Anonymous Robots. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, Lecture Notes in Computer Science, pages 109–133. Springer International Publishing, Cham, 2019. ISBN 9783030110727. doi: 10.1007/978-3-030-11072-7_6. URL `https://doi.org/10.1007/978-3-030-11072-7_6`.

[80] Keisuke Doi, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Exploration of Finite 2D Square Grid by a Metamorphic Robotic System. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems*, Lecture Notes in Computer Science, pages 96–110, Cham, 2018. Springer International Publishing. ISBN 9783030032326. doi: 10.1007/978-3-030-03232-6_7.

[81] Ryonosuke Yamada and Yukiko Yamauchi. Search by a Metamorphic Robotic System in a Finite 3D Cubic Grid: 1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022. *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022*, April 2022. doi: 10.4230/LIPIcs.SAND.2022.20. URL http://www.scopus.com/inward/record.url?scp=85130775621&partnerID=8YFLogxK.

[82] Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of Cube-Style Modular Robots Using O(logn) Parallel Moves. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, Lecture Notes in Computer Science, pages 342–353, Berlin, Heidelberg, 2008. Springer. ISBN 9783540921820. doi: 10.1007/978-3-540-92182-0_32.

[83] Matthew Connor, Othon Michail, and Igor Potapov. Centralised connectivity-preserving transformations for programmable matter: A minimal seed approach. In *17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 45–60. Springer, 2021. doi: https://doi.org/10.1007/978-3-030-89240-1_4. URL https://link.springer.com/chapter/10.1007/978-3-030-89240-1_4.

[84] Matthew Connor, Othon Michail, and Igor Potapov. Centralised connectivity-preserving transformations for programmable matter: A minimal seed approach. *Theoretical Computer Science*, 936:77–91, 2022. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2022.09.016. URL https://www.sciencedirect.com/science/article/pii/S0304397522005527.

[85] Matthew Connor and Othon Michail. Centralised Connectivity-Preserving Transformations by Rotation: 3 Musketeers for All Orthogonal Convex Shapes. In Thomas Erlebach and Michael Segal, editors, *Algorithmics of Wireless Networks*, Lecture Notes in Computer Science, pages 60–76, Cham, 2022. Springer International Publishing. ISBN 9783031220500. doi: 10.1007/978-3-031-22050-0_5.

[86] Camille Jordan. *Cours d'analyse de l'École polytechnique*, volume 1. Gauthier-Villars et fils, 1893.

[87] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2460–2467 vol.3, October 2003. doi: 10.1109/IROS.2003.1249239.

[88] Cynthia Sung, James Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1933–1940, May 2015. doi: 10.1109/ICRA.2015.7139451. ISSN: 1050-4729.

[89] Cem Ünsal, Han Kiliççöte, and Pradeep K. Khosla. A Modular Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning. *Autonomous Robots*, 10(1):23–40, January 2001. ISSN 0929-5593. doi: 10.1023/A:1026592302259. URL `https://doi.org/10.1023/A:1026592302259`.