Shape Recognition by a Finite Automaton Robot

Robert Gmyr
Paderborn University, Germany
gmyr@mail.upb.de

Kristian Hinnenthal
Paderborn University, Germany
krijan@mail.upb.de

Irina Kostitsyna
TU Eindhoven, the Netherlands
i.kostitsyna@tue.nl

Fabian Kuhn
University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Dorian Rudolph
Paderborn University, Germany
dorian@mail.upb.de

Christian Scheideler
Paderborn University, Germany
scheideler@upb.de

Abstract

Motivated by the problem of shape recognition by nanoscale computing agents, we investigate the problem of detecting the geometric shape of a structure composed of hexagonal tiles by a finite-state automaton robot. In particular, in this paper we consider the question of recognizing whether the tiles are assembled into a parallelogram whose longer side has length $\ell = f(h)$, for a given function $f(\cdot)$, where h is the length of the shorter side. To determine the computational power of the finite-state automaton robot, we identify functions that can or cannot be decided when the robot is given a certain number of pebbles. We show that the robot can decide whether $\ell = ah + b$ for constant integers a and b without any pebbles, but cannot detect whether $\ell = f(h)$ for any function $f(x) = \omega(x)$. For a robot with a single pebble, we present an algorithm to decide whether $\ell = p(h)$ for a given polynomial $p(\cdot)$ of constant degree. We contrast this result by showing that, for any constant k, any function $f(x) = \omega(x^{6k+2})$ cannot be decided by a robot with k states and a single pebble. We further present exponential functions that can be decided using two pebbles. Finally, we present a family of functions $f_n(\cdot)$ such that the robot needs more than n pebbles to decide whether $\ell = f_n(h)$.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Design and analysis of algorithms

Keywords and phrases finite automata, shape recognition, computational geometry

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.52

Funding This work is partly supported by DFG grant SCHE 1592/3-1. Fabian Kuhn is supported by ERC Grant 336495 (ACDC).

Acknowledgements This work was begun at the Dagstuhl Seminar on Algorithmic Foundations of Programmable Matter, July 3–8, 2016. Preliminary results were presented at EuroCG 2018.

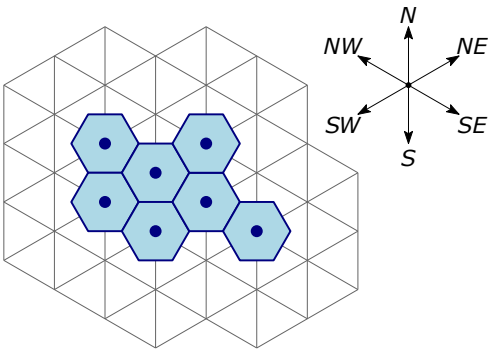

Figure 1 An exemplary tile configuration. The top right part of the figure shows the compass directions we use to describe the movement of a robot.

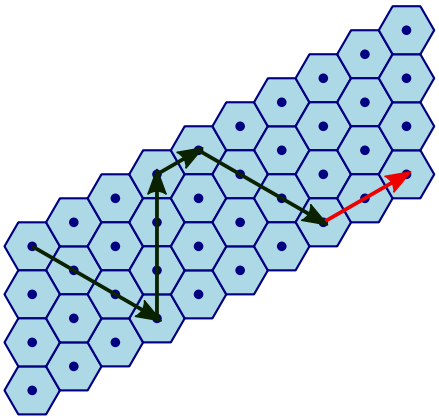

Figure 2 A N-NE-parallelogram with height 4 and length $10 = 2 \cdot 4 + 2$. The black arrows indicate the zig-zag movement of a robot as described in the proof of Theorem 2. The red arrow shows the final NE movement.

1 Introduction

The *DNA-based self-assembly* approach [20], in which large structures have to be assembled from DNA tiles in a self-organized fashion, is still quite error prone. Numerous techniques have been studied to reduce the error rates (e.g., [6]), but the research typically focuses on designing DNA tiles or assembly processes that reduce the error of incorrect attachments. Given the progress on designing so-called *DNA walkers* (see, e.g., [22, 24, 25]), it is foreseeable that also simple molecular robots could be used in order to make the assemblies more reliable. For example, such robots may be used to repair structures that did not self-assemble correctly. In order to be able to repair a given structure, it first has to be possible to detect whether a structure is not in a correct shape, which motivates the problem of *shape recognition*. Besides verifying self-assembled DNA structures, shape recognition could also be useful for minimal invasive surgery applications, such as looking for harmful substances or cells in a human body.

Naturally, molecular robots may have very limited capabilities, which is why we are focusing on robots with the computational power of a finite-state automaton. More precisely, we build upon the model introduced in [10] where finite-state automaton robots move on a structure composed of hexagonal tiles, represented as a subgraph of the infinite triangular lattice, and rearrange the tiles into a certain shape. Although shape *formation* with computationally restricted agents, as considered in [10], has been extensively studied in many other models (see, e.g., [5, 13, 17, 26]), to the best of our knowledge, the closely related problem of shape *detection* has never been explicitly studied in our setting.

1.1 Model

We assume that a single *robot* is placed on a finite set of *hexagonal tiles*. Each tile occupies exactly one node of the infinite triangular lattice $G = (V, E)$ (refer to Figure 1). We assume that the subgraph of G induced by all nodes occupied by tiles is connected. Every node $u \in V$ is adjacent to six neighbors, and, as indicated in the figure, we describe the relative positions of adjacent nodes by six compass directions.

Table 1 This table summarizes the results for recognizing whether a given parallelogram has height h and length $\ell = f(h)$ given a certain number of pebbles.

Pebbles	Possible	Impossible	Remarks	Refer to
0	$f(x) = ax + b$	$f(x) = \omega(x)$	a, b constant	Section 3.1
1	$f(x) = a_n x^n + \ldots + a_0$	$f(x) = \omega(x^{6k+2})$	n, a_i constant for all i, k is the number of the robot's states	Section 3.2
2	$f(x) = \underbrace{2^{2^{\cdot^{\cdot^{2^x}}}}}_{s+1}$	—	s constant	Section 3.3
n	$f_n(x)$	$f_{n+1}(x)$	—	Section 3.4

The robot is initially placed on a tile. It can move on the tile structure and carry a (possibly empty) set of *pebbles*, which can be placed on tiles in order to *mark* them. A tile can be marked by at most one pebble.

More specifically, the robot acts as a *deterministic finite automaton* and operates in *look-compute-move* cycles. In the *look* phase the robot can observe the node it occupies and the six neighbors of that node. For each of these nodes it can determine whether it is occupied by a tile and whether a pebble is placed on that tile. In the *compute* phase the robot potentially changes its state and determines its next move according to the observed information and the number of carried pebbles. In the *move* phase the robot can either take a pebble from its current node (if its tile is marked by a pebble), place a pebble it is carrying at that node (if its tile is not already marked and the robot carries at least one pebble), or move to an adjacent occupied node.

Note that even though we describe the algorithms as if the robot knew a global orientation, we do not actually require the robot to have a compass. For the algorithms presented in this paper, it is enough for the robot to be able to maintain its orientation with respect to its initial orientation.

1.2 Related Work

To the best of our knowledge, shape recognition has never been investigated in our model. However, solving problems by traversing a tile structure with simple agents has been studied in many different areas. For instance, [23] considers the problem of deciding whether a structure is simply-connected. Other problems include Gathering and Rendezvous (e.g., [21]), Intruder Caption and Graph Searching (e.g., [2,8]), or Black Hole Search (e.g., [16]).

For many of the above problems it has also been investigated whether pebbles can be helpful. This question is particularly well-studied for the classical Network Exploration Problem (see, e.g., [4]). For example, it is known that a finite automaton robot can neither explore all planar graphs [9] nor find its way out of a planar labyrinth [3] without any pebbles. For the Labyrinth Exploration Problem (see [14,15] for a comprehensive survey), it is known that having a single pebble does not help the robot [11]. However, a robot with two pebbles can solve the problem [1].

1.3 Our Contributions

In this paper we investigate the problem of *shape recognition* by a single robot. Specifically, we begin with testing whether a given tile formation is of a certain simple shape, in particular, a parallelogram. Then, we consider the problem of deciding for a given function $f(\cdot)$ whether the longer side of a parallelogram has length $f(h)$, where h is the shorter side's length. We particularly investigate this problem under the assumption that the robot is given a set of pebbles that can be used to mark certain positions on the tile structure. An overview of our results is given in Table 1, in which we give concrete functions $f(\cdot)$ the robot is able or not able to decide given a certain set of pebbles. Our ultimate goal is to investigate the computational capabilities of a simple robot concerning shape recognition, and to what extent the robot can benefit from employing pebbles.

As we do not make any assumptions on the length of the shorter side h of the parallelogram, we cannot assume that a robot with only a constant number of possible states is able to count up to h, even less so evaluate the function $f(h)$. Thus, if the robot does not have pebbles at its disposal, its only option is to make use of the environment's geometry. For example, the robot can 'measure' h tiles along the longer side of the parallelogram by starting in a corner and moving diagonally until reaching the opposite boundary. Furthermore, the robot can measure $2h$, $3h$, or even ah tiles, for any constant a, along the longer side.

In Section 3.1 we develop this intuition further, and show that the robot can decide whether the longest side of the parallelogram is $\ell = ah + b$, where a and b are constants. On the other hand, we show that the robot without pebbles is not able to recognize a superlinear function. In Sections 3.2 and 3.3 we show that we can tremendously increase the robot's computational power by giving it a single pebble or two pebbles, respectively. Having the ability to mark any tile with a single pebble allows the robot to recognize any polynomial function of constant degree; and being equipped with two pebbles gives the robot the ability to recognize power tower functions, where the height of the power tower is constant or even linear in h. Finally, in Section 3.4 we show that for any number of pebbles there exists a function that requires that many pebbles to be decided by the robot.

2 Recognizing Simple Shapes

First, observe that a single robot can easily detect whether the initial structure is a line, a triangle, a hexagon, or a parallelogram.

Line. For example, to test if a given tile shape is a line, the robot first chooses a direction in which there is a tile (say, w.l.o.g., N), walks in that direction as far as possible (i.e., until there is no tile in that direction anymore), and then traverses the structure into the opposite direction until no longer possible. If it ever encounters a tile to the left or right of any traversed tile, the structure is not a line.

Parallelogram. To test if a given tile shape is a (filled) parallelogram axis-aligned along the directions N and NE (see Figure 2), first, the robot moves to a locally southernmost tile of the structure by moving S and SW as long as there is a tile in any of these directions. It then traverses the shape column by column in a snake-like fashion by repeating the following movements: First, the robot moves N as far as possible; it then moves one step NE; then it moves S as far as possible, and it finally moves one step NE. The above procedure is repeated until a NE movement is impossible. By performing local checks alongside the movements described above the robot can verify whether the tile shape is a parallelogram.

The other simple shapes can be easily tested in a similar fashion.

▶ **Observation 1.** *A robot without any pebble can detect whether the initial tile configuration is a line, a triangle, a hexagon, or a parallelogram.*

3 Recognizing Parallelograms with Specific Side Ratio

As noted in Observation 1, a single robot without pebbles can verify whether a given shape is a parallelogram. To investigate the computational power of a finite automaton, in this section we consider the problem of deciding whether a parallelogram has a given side ratio. Additionally, we examine how pebbles can be helpful to decide more complex side ratios.

We assume w.l.o.g. that the robot needs to detect whether the given tile configuration is a parallelogram that is axis-aligned along the north and north-east direction, as indicated in Figure 2. We denote a maximal sequence of consecutive tiles from N to S as a *column* and a maximal sequence of consecutive tiles from SW to NE as a *row*. Let h be the size of each column, i.e., the parallelogram's *height*, ℓ be the size of each row, i.e., the parallelogram's *length*, and let $h \leq \ell$. We number the columns of the parallelogram from 0 to $\ell - 1$ growing in the north-eastern direction.

3.1 A Robot without any Pebble

First, we point out that a single robot can detect whether the structure is a parallelogram in which its length ℓ is a linear function of its height h.

▶ **Theorem 2.** *A single robot can detect whether the tile configuration is a parallelogram with $\ell = ah + b$ for any constants $a, b \in \mathbb{N}$.*

Proof. First, the robot verifies whether the structure is a parallelogram. If so, the robot moves to the northernmost tile of column 0. It then traverses the tile structure in two stages to verify the ratio of the sides. In the first stage, the robot "measures" the distance ah along the length of the parallelogram moving in a zig-zag fashion as depicted in Figure 2. In the second stage the robot measures the second term b. More specifically, in the first stage, the robot repeats the following movements in a loop: (1) move SE as far as possible, (2) move N as far as possible, and (3) make one step NE. After having performed the complete sequence of SE movements a times, the robot moves on to the second stage, in which it makes an additional b NE steps.

If the robot reaches the easternmost column before completing the above procedure, or finally halts on a tile with a neighboring tile at NE, it terminates with a negative result. Otherwise, it terminates with a positive result. It is easy to see that $\ell = ah + b$ if and only if the robot terminates with a positive result. ◀

▶ Remark. The algorithm in the previous theorem can be adapted for $b \in \mathbb{Z}$, i.e., b can also be a negative integer. To achieve that, we halt the execution of the first stage once the robot has performed $|b|$ SE steps, then move SW as far as possible, and continue the first stage from there. Then, $\ell = ah + b$ if and only if the robot eventually reaches the southernmost tile of column $\ell - 1$.

▶ Remark. The algorithm can be further extended to apply in the case of a rational a. Let $a = p/q$ be an irreducible fraction. Instead of moving in a zig-zag fashion in the first stage, the robot alternates between moving p steps NE and q steps S. To exactly end up at the southernmost tile of column $\ell - 1$, the robot needs to skip the very first NE and S step.

We have shown that a single robot can determine whether the length of a parallelogram is given by a certain linear function of its height. However, that is as much as one robot can hope for. Indeed, a single robot is not able to decide whether the length of the parallelogram is given by a superlinear function of its height, as the following theorem states.

▶ **Theorem 3.** *A single robot without any pebbles cannot decide whether the tile configuration is a parallelogram with $\ell = f(h)$, where $f(x) = \omega(x)$.*

Proof. Suppose there is an algorithm that lets the robot decide whether the tiles are arranged into a parallelogram with $\ell = f(h)$ for some superlinear function $f(x) = \omega(x)$. Let k be the total number of states used by the robot in the algorithm. Choose h large enough such that $\lfloor \frac{f(h)-2}{h} \rfloor > k$, which can be done since $f(h) = \omega(h)$.

Consider the execution of the algorithm on a parallelogram P with height h and length $\ell = f(h)$. We will show that there exists another parallelogram with height h and length greater than $f(h)$ on which the robot will eventually terminate in exactly the same state as on P, which contradicts the assumption that the algorithm is correct.

We first place the robot on the northernmost tile of column 0. First, observe that if the robot does not visit column $\ell - 1$ during the execution of the algorithm, it cannot correctly detect the length of the parallelogram. Thus, the robot visits this column at least once.

Consider the execution of the algorithm as a sequence $(p_1, p_2, \dots)$ of tuples of nodes and states $p_i = (u_i, q_i)$. Denote as $\pi_1, \dots, \pi_m$ the subsequences of the execution of the algorithm, where each subsequence starts whenever the robot leaves a node of column 0 or $\ell - 1$, and ends when it enters a node of one of those columns. More specifically, the first and last node of each π_i is a node adjacent to column 0 or $\ell - 1$ and the node immediately before and after each π_i is a node of 0 or $\ell - 1$. Note that in each π_i the robot exclusively moves in the columns between column 0 and $\ell - 1$.

First, consider a subsequence π_i at the beginning of which the robot leaves, and at the end of which, enters the same column 0 (or $\ell - 1$). Let $p_i = (s_i, q_i)$ be the node-state tuple right before the robot enters the subsequence π_i when executing the algorithm on the parallelogram P. Then, the robot would execute exactly the same subsequence π_i on a parallelogram P' with a larger length than the one of P, if it were placed on a node of P' corresponding to s_i with the same state q_i.

Next, consider a subsequence π_j at the beginning of which the robot leaves, w.l.o.g., column 0, and at the end of which it enters $\ell - 1$. Since the robot completely traverses the tile structure from column 0 to $\ell - 1$, there must be a row R_j in which the robot steps on no less than $\lfloor \frac{f(h)-2}{h} \rfloor > k$ tiles. Therefore, there will be two nodes u_j and v_j in the row R_j in which the robot appears in the same state. Let c_u, c_v be the column indices of u_j and v_j, respectively, and define $d_j = |c_v - c_u|$. Let (s_j, q_j) be the node-state tuple of the robot immediately before it enters the subsequence π_j and (s'_j, q'_j) be the node-state tuple of the robot immediately after the subsequence π_j is finished. Then, consider a parallelogram Q with height h and length $f(h) + cd_j$, where $c \in \mathbb{N}_0$. If the robot starts in the same state q_j on the node corresponding to s_j (i.e., the node in column 0 and the row of s_j) in the parallelogram Q, it will move entirely between the westernmost and easternmost column of Q until it reaches the node corresponding to s'_j (i.e., the node of column $f(h) + cd_j$ and the row of s'_j) in Q in state q'_j.

Now consider the execution of the algorithm on a parallelogram P' with height h and length $f(h) + \prod_j d_j$ for each subsequence π_j where the robot completely traverses the parallelogram between column 0 and $\ell - 1$. By the above argument, the robot will enter and leave those columns on the same tile and in the same state as in the execution of the algorithm on P. Thus, executing the same algorithm on the parallelogram P' the robot will ultimately terminate in the same state as if it were on the parallelogram P. Therefore, the robot cannot decide whether the initial tile configuration is a parallelogram with $\ell = f(h)$ for any superlinear function $f(x) = \omega(x)$. ◀

3.2 A Robot with a Single Pebble

In the following, we demonstrate that, in contrast to the negative result of Theorem 3, a single robot can decide any polynomial of constant degree.

▶ **Theorem 4.** *A single robot with a pebble can decide whether the tile configuration is a parallelogram of height h and length $\ell = p(h)$ for any given polynomial $p(\cdot)$ of constant degree n.*

Proof. Define the *falling factorial* of x as $(x)_i := x(x-1)\cdots(x-i+1)$, and transform the input polynomial into the form $p(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_0$. We will show that the robot can move the pebble in phases, by $|a_i \cdot (h)_i|$ steps in each phase i. Let $\mathrm{lcm}_i(x) := \mathrm{lcm}(x, \ldots, x-i+1)$, where lcm is the *least common multiple*, and $g_i(x) := (x)_i / \mathrm{lcm}_i(x)$. From [12] it follows that $\mathrm{lcm}_i(x) \mid (x)_i$, and that $g_i(x)$ is periodic with period $\mathrm{lcm}(1, \ldots, i-1)$, i.e., $g_i(x) = g_i(x + \mathrm{lcm}(1, \ldots, i-1))$. Let $p_i(x)$ be the sum of the first $n-i$ summands of $p(x)$, i.e., $p_i(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_{n-i+1} \cdot (x)_{n-i+1}$.

Initially, the pebble is located on the northernmost tile of column 0. To test whether $\ell = p(h)$, the robot will move the pebble along the northernmost row in phases, until it is eventually shifted $p(h) - 1$ steps to the *NE* from its original position. If upon termination the pebble is located at the northernmost tile of column $\ell - 1$, then $p(h) = \ell$.

The algorithm proceeds in phases $n, \ldots, 0$. We maintain the invariant that after phase i for all $i > 0$, the pebble is located at the northernmost tile of column $p_i(h)$. That is, in phase i, the robot moves the pebble $|a_i \cdot (h)_i|$ steps *NE*, if a_i is positive, and *SW*, otherwise. In the final phase $i = 0$, the robot moves the pebble by $|a_0 - 1|$ steps *NE*, if $a_0 \geq 1$, and *SW*, otherwise. For now, assume that each movement can be carried out without moving the pebble outside of the parallelogram. We will later describe how to lift this restriction.

We now describe how the pebble is moved by $|a_i \cdot (h)_i|$ steps. First, note that $a_i \cdot (h)_i = a_i \cdot g_i(h) \cdot \mathrm{lcm}_i(h)$. The first factor a_i is a constant. The second factor $g_i(h)$ can be determined as follows. We encode all possible values of $g_i(\cdot)$ for all $i \in \{0, \ldots, n\}$ into the robot's memory, which can be done since n is constant and $g_i(\cdot)$ has a constant period. Before the main algorithm's execution, the robot can compute $g_i(h)$ for all i by moving through column 0 from north to south: Starting with $g_i(1)$, in every step to the south the robot computes the subsequent function value until the period of $g_i(\cdot)$ is reached, in which case it restarts with $g_i(1)$. When it reaches the southernmost tile of the column, it knows $g_i(h)$ for all i.

We next show how the robot moves the pebble by $\mathrm{lcm}_i(h)$ steps, which, by repeating the movement $|a_i \cdot g_i(h)|$ times, concludes how the complete movement by $|a_i \cdot (h)_i|$ steps is performed. Assume the pebble is in some column c and $\mathrm{lcm}_i(h) \mid c$ (which we will prove by induction shortly). The robot alternates between the following two operations: (1) move the pebble into column c' by moving it one step *NE*, if $a_i > 0$, or *SE*, otherwise; (2) verify whether $\mathrm{lcm}_i(h) \mid c'$ as follows. The robot first performs the zig-zag movement from the proof of Theorem 2 to verify whether $h \mid c'$, i.e., whether a *NE* movement moves the robot onto a tile occupied by the pebble. It continues to analogously verify whether $h-1 \mid c'$, $h-2 \mid c'$, $\ldots$, $h-i+1 \mid c'$ by performing a modified zig-zag movement an additional $i-1$ times. Here, the zig-zags of the j-th verification are adjusted accordingly by moving j steps to the south prior to each sequence of *SE* movements. The robot stops alternating between the two above operations once the pebble has been moved to a column c' such that $\mathrm{lcm}_i(h) \mid c'$ for the first time. Then, the pebble must have been moved by $\mathrm{lcm}_i(h)$ steps.

It remains to prove that when the robot wants to move the pebble, currently occupying a node of column c, by $\mathrm{lcm}_i(h)$ steps for some i, then $\mathrm{lcm}_i(h) \mid c$. The invariant holds initially for $c = 0$. Now assume it holds immediately after having moved the pebble by $\mathrm{lcm}_i(h)$ steps

into column $c \pm \mathrm{lcm}_i(h)$. By induction hypothesis, $\mathrm{lcm}_i(h) \mid c$. Afterwards, the robot can move the pebble by either $\mathrm{lcm}_i(h)$ steps again, in which case $\mathrm{lcm}_i(h) \mid c \pm \mathrm{lcm}_i(h)$ holds, or it moves it by $\mathrm{lcm}_{i-1}(h)$ steps, and $\mathrm{lcm}_{i-1}(h) \mid c \pm \mathrm{lcm}_i(h)$ holds since $\mathrm{lcm}_{i-1}(h) \mid \mathrm{lcm}_i(h)$.

Finally, we show how the robot can resolve *overflows*, i.e., situations in which the above algorithm would move the pebble outside of the parallelogram. First, note that the execution of the algorithm after an overflow can, in principle, be continued by the robot by "mirroring" all movements beyond the westernmost or easternmost column, carrying them out into reverse direction. Assume that h is sufficiently large such that $|a_i \cdot (h)_i + \ldots + a_0| \leq p(h)$ for all i. For all small $h = O(\max_i(|a_i|))$ we can encode the constantly many possible function values into the robot's state and test them prior to the algorithm's execution by traversing the two sides of the parallelogram once. If throughout the execution of the algorithm the robot ever attempts to move the pebble into a column west of column 0 or east of "virtual" column 2ℓ (while performing the mirroring method from above), it would subsequently not be able to ever move the pebble back into column ℓ (following from the assumption that h is sufficiently large), and consequently $\ell \neq p(h)$. Therefore, the robot can prematurely terminate with a negative result whenever it encounters such a situation. ◀

The next theorem gives a lower bound on the amount of states needed to decide whether $\ell = h^a$, $a \in \mathbb{N}$, thereby proving that no robot with one pebble can decide whether $\ell = f(h)$ for $f(x) = \omega(x^a)\, \forall a \in \mathbb{N}$.

▶ **Theorem 5.** *A robot with k states and a single pebble cannot decide whether the tile configuration is a parallelogram of height h and length $\ell = f(h)$, $f(x) = \omega(x^{6k+2})$.*

Proof. First, we give a brief outline of the following proof. Assuming such a robot exists, we place the robot with its pebble on the northernmost tile of column 0 of a parallelogram P with height h and length $\ell = f(h)$. We begin by subdividing P horizontally into parallelograms of height h which we will refer to as *blocks*. We define the westernmost and easternmost blocks as the *outer blocks* O_ℓ and O_r, respectively, and denote all other blocks as *inner blocks*. We choose the length b of all inner blocks such that when the robot moves through a sequence of inner blocks (from NW to SE or SE to NW), while either carrying the pebble the entire time or not visiting it at all, its row and state repeat every b columns. The outer blocks will have length at least b.

As in the proof of Theorem 3, we consider the execution of the algorithm as a sequence of tuples of nodes and states, and divide it into subsequences $\pi_1, \ldots, \pi_m$. Subsequence π_i starts with the robot carrying the pebble into some outer block, and ends when it reaches the opposite outer block while carrying the pebble. The robot terminates in π_m, and, as this is the final subsequence, before entering the opposite block while carrying the pebble. We define r_{π_i} and q_{π_i} to be the robot's row and state at the beginning of π_i. By considering where the robot places and picks up the pebble, we identify a value d such that r_{π_i} and q_{π_i} remain the same if the robot is executed on a parallelogram P' of height h and length $f(h) + db$, and therefore falsely terminates with a positive result.

We begin the proof by identifying a value b. Consider the robot's execution without the pebble on a parallelogram of height h and infinite length in both directions, starting in row r and state q. If the robot moves by at most kh columns in NW or SE direction, we define $d_{r,q} := 1$. Otherwise, by the pigeonhole principle, there are two columns that are visited on a node of the same row and in the same state. In this case, we define $d_{r,q}$ to be the distance between these two columns. Note that the robot moves arbitrarily far in this case, its row and state repeating every $d_{r,q}$ columns. Let $D := \{d_{r,q}\}$. Analogously, we consider the execution

of the robot if it initially carries the pebble, and define $d^*_{r,q} := 1$, if it ever drops the pebble or moves by at most kh columns in NW or SE direction. Otherwise, there is a repetition every $d^*_{r,q}$ columns, and the robot carries the pebble indefinitely far. Let $D^* := \{d^*_{r,q}\}$.

We now show that $|D| \leq 3k$. If the robot ever is in row r in state q, or row r' in state q', and visits the northernmost (or southernmost) row in the same state q^* in both executions, the executions will be identical afterwards, and therefore, $d_{r,q} = d_{r',q'}$. Hence, there can only be k combinations of rows and states with distinct executions in which the robot visits the northern- or southernmost row, and these executions contribute at most $2k$ distinct distances to D. Now, let q, r, r' such that the robot does not visit the northern- and southernmost row when started in state q in row r or r'. Clearly, the robot performs the exact same actions in both executions. Therefore, $d_{r,q} = d_{r',q}$, and these executions contribute at most k additional distances. Therefore, $|D| \leq 3k$. Analogously, we have $|D^*| \leq 3k$.

We set $b := \prod_{\delta \in D \cup D^*} \delta$. If $b \leq kh$, redefine $b := (kh + 1)b$. It holds that $b = O(h^{6k})$. We subdivide P into a maximum number of blocks such that inner blocks have length b and outer blocks have length greater than b.

We now identify a value d. Consider the subsequence π_i, $i < m$. W.l.o.g., assume the robot moves the pebble from O_ℓ to O_r. Let (r_j, c_j, q_j), $j = 1, \ldots, l$ be the row, column and state in which the pebble is dropped during π_i. We distinguish two cases: In the first case, the robot moves by more than kh columns while carrying the pebble. In this case, it will move until reaching the easternmost column C_r due to having a repetition in its state and row. Therefore, $c_{j+1} - c_j \leq kh$ for $j < l$. In this case, set $d_i = 1$.

In the second case, the robot does never move by more than kh columns to the east while carrying the pebble during π_i. Thus, the pebble is dropped within the first (i.e., western) kh columns of each inner block. P contains $\Omega(f(h)/b) = \omega(h^2) > (kh)^2$ inner blocks, for sufficiently large h. For some column c, we denote its index inside its block as $\tilde{c}$. Hence, for each inner block, there exists a j such that c_j is in that block and $\tilde{c}_j < kh$. There are at most $h \cdot kh \cdot k = (kh)^2$ possibilities for $(r_j, \tilde{c}_j, q_j)$. By the pigeonhole principle, there exist $s < t$ such that $c_s < c_t$ and $(r_s, \tilde{c}_s, q_s) = (r_t, \tilde{c}_t, q_t)$. We set $d_i := (c_t - c_s)/b$, i.e., the number of inner blocks between c_s and c_t, and $d := \prod_{i=1}^{m} d_i$.

Let P' be the parallelogram of height h and length $f(h) + db$. Now we show that $(r_{\pi_{i+1}}, q_{\pi_{i+1}})$ is the same in the execution on P and P' for all $i < m$ and that the robot terminates with the same result during π_m. To that end, we will again look at a subsequence π_i, $i < m$ where, w.l.o.g, the pebble is carried from O_ℓ to O_r and compare the executions on P and P'. Let (r_j, c_j, q_j), $j = 1, \ldots, l$ as above. The first l times the pebble is dropped on P' are identical to those on P, since the only difference in the execution between dropping the pebble at a column c_j, $j \leq l$ and picking it back up again can be when the robot moves to column $\ell - 1$. As argued above, moving to column $\ell - 1$ on P and P' cannot be distinguished by the robot due to a repetition in row and state and by our choice of b. Thus, the pebble is picked up in the same state again.

Next, we need to look at what happens on P' after the pebble has been picked up for the l-th time. Here, we distinguish between the two cases from above. In the first case, the robot carries the pebble by more than kh columns to the east on P before entering O_r and, consequently, continues until reaching column $\ell - 1$. The same behavior occurs on P', only that the robot traverses an additional d blocks and, by our choice of b, enters O_r in the same row and state as on P.

In the second case, it never carries the pebble by more than kh columns to the east during π_i. We identified s, t such that $(r_s, \tilde{c}_s, q_s) = (r_t, \tilde{c}_t, q_t)$, i.e. the pebble is picked up in the same row, block-column, and state. Note that whenever the robot drops the pebble in some

column $\tilde{c}$ of two different inner blocks, it will pick the pebble up again in the same state: This is clearly true if the robot does not visit column 0 and $\ell - 1$ between dropping and placing the pebble. Otherwise, the robot traverses more than kh columns without seeing the pebble, i.e. its row and state repeat every δ columns for some $\delta \in D$. Consequently, by our choice of b, after both drops the robot will return to the pebble (and pick it up) in the same state. Therefore, in both the executions on P and P' the robot will repeatedly drop and pick up the pebble, each repetition occurring d_i blocks to the east from the previous one. As $d_i \mid d$, the robot enters O_r in the same row on P and P' and we thus also have identical $(r_{\pi_{i+1}}, q_{\pi_{i+1}})$.

It only remains to show that the robot terminates with the same result during π_m. W.l.o.g, let π_m begin in O_l on P, and recall that the pebble does not reach O_r before the robot terminates. As argued before, the executions on P and P' can only differ when the robot drops the pebble and moves to the easternmost column of the respective parallelogram. Due to a repetition in state and row, and by our choice of b, the robot again enters the respective easternmost columns in the same state and row, and afterwards picks up the pebble in the same state on both P and P'. Therefore, the robot ultimately terminates in the same state. ◀

3.3 A Robot With Two Pebbles

Next, we show that having two pebbles enables the robot to decide certain exponential functions. Note that by Theorem 5, the following result is optimal in the number of pebbles used.

▶ **Theorem 6.** *A robot with two pebbles can decide whether a given tile configuration is a parallelogram with height $h > 1$ and length*

$\ell = 2^{2^{\cdot^{\cdot^{2^h}}}}$ *, where the power tower is of constant height.*

Proof. Let $s + 1$ be the height of the power tower (i.e., there are s twos in the function) and denote the two pebbles as a and b. Pebble a always resides in the northernmost row of the parallelogram. We use a's column index as a register on which we perform basic arithmetic operations using b as a helper. Note that although the easternmost column of the parallelogram has index $\ell - 1$, we describe the algorithm as if there was an additional column ℓ. A movement from or to column ℓ can easily be simulated by the robot.

The algorithm is divided into three *stages*. The purpose of the first stage is to verify that ℓ is a power of 2. In the second stage, we move a from column ℓ to column $\log^{(s-1)}(\ell)$, where $\log^{(s-1)}(\cdot)$ denotes the application of the logarithm $s - 1$ times. In the third stage, we verify that a is in column 2^h after the second stage. If in any of the three stages the robot detects a violation of any assumption, i.e., if according to the algorithm a would have to be moved beyond column 0 or ℓ, or should be in column 0 or ℓ, but is not, the robot terminates with a negative result.

Before we describe the three stages in more detail, we describe how a's column index can be multiplied or divided by any constant $c \geq 1$, provided that the result is integer and between 0 and ℓ. We first place b at a tile south of a, and then move a into column 0. In case of a multiplication, we then alternatingly move a *NE* by c steps, and b *SE* by one step, until b reaches column 0. In case of a division, we correspondingly move b by c steps and a by one step.

In the first stage, the robot does the following. It first places a at the northernmost tile of column 1. It then repeatedly multiplies by 2 (i.e., multiplies a's column index by 2) using

the above-mentioned strategy. If b reaches column 0 immediately after a has reached column ℓ, ℓ is a power of 2.

At the beginning of the second stage, a is placed at the northernmost tile of column ℓ. The stage is divided into $s-1$ *phases*, where in each phase a is moved from column i to column $\log(i)$. Note that since s is a constant, the robot can count the number of phases. Furthermore, after the first phase ℓ is verified to be a power of 2, and, as we will show later, if a's column index is 2^x at the beginning of a phase, but x is not a power of 2, then the phase fails. Therefore, a's column index is ensured to be a power of 2 at the beginning of each phase.

In each phase, a is moved from some column 2^x to column x in a step-wise fashion. More precisely, in the j-th step it is moved from column $5^{j-1} \cdot 2^{x/2^{j-1}}$ to column $5^j \cdot 2^{x/2^j}$. This is continued until the resulting column index is not divisible by 4 anymore, i.e., when it becomes $5^{\log x} \cdot 2$. The general idea is to use the exponent of 5 as a counter on the number of times x needs to be divided by 2 until it becomes 1, which yields its logarithm. This idea is based on [7, Section 14].

It remains to show how a single step can be performed. First, the robot moves a from column $5^{j-1} \cdot 2^{x/2^{j-1}}$ into column $5^{j-1} \cdot 3^{x/2^j}$ by alternatingly dividing by 4 and multiplying by 3. After each repetition, the robot verifies whether a's column index is divisible by 2. This is done by traversing the northernmost row from west to east until a is reached, and counting the number of steps modulo 2. Note that if $x/2^{j-1}$ is even, which is the case if x is a power of 2, then the division by 4 is always possible. Otherwise, the division by 4 will fail at latest when $x/2^{j-1}$ becomes 1. Once a's column index is not divisible by 2 anymore, a is in column $5^{j-1} \cdot 3^{x/2^j}$.

Analogously, by alternatingly dividing a's column index by 3 and multiplying by 2 as long as the column index is divisible by 3, the robot afterwards moves a into column $5^{j-1} \cdot 2^{x/2^j}$. By multiplying with 5, a is finally moved into column $5^j \cdot 2^{x/2^j}$.

After each step, the robot verifies whether a's column index is divisible by 4. If so, it continues with the next step. Otherwise, $j = \log x$, and thus a must be in column $5^{\log x} \cdot 2$. From there, a can easily be moved into column x by first dividing by 2, and afterwards alternatingly dividing by 5 and multiplying by 2 until the column index is not divisible by 5 anymore.

Note that if a's column index is 2^x at the beginning of some phase, but x is not a power of 2, then at some step $x/2^{j-1}$ will be odd, and, as described above, the algorithm will fail. Further note that for $h \geq 8$, which can be verified beforehand, moving a from column 2^x (where, consequently, x must be at least 8) to $5^{\log x} \cdot 2$ can only decrease a's column index. Therefore, although a might be moved *NE* in the final steps of a phase, it can easily be seen it will be moved sufficiently far *SW* within the first steps such that it is never moved beyond column ℓ. If that happens nonetheless, the robot terminates with a negative result.

Finally, in the third stage, the robot verifies that a is in column 2^h by diving by 2 for h times. To count up to h, b initially resides in row 2 and is moved one step S after each division. When b reaches a southernmost tile, a must lie in column 4, which can easily be verified by the robot. ◀

▶ Remark. The algorithm of the previous theorem can be adapted for any power towers whose height is a linear function $\alpha h + \beta$ for constants α and β. To count the number of phases in the second stage, we move a south after each phase. The highest exponent of the power tower may also be a function linear in h, which can be handled correspondingly in the third stage.

▶ Remark. The algorithm can further be adapted for any other base β. Note that if β is a composite number, we need to apply the operations to its prime factors separately, taking into account their powers. Since it is well-known that for $n \geq 25$ there is always a prime between n and $(1 + 1/5)n$ [19], we can use the two smallest primes that are no prime factors of β instead of 3 and 5 in the second stage of the algorithm.

▶ Remark. In contrast to the negative results of the previous sections, it can be shown that for every computable function f there exists a computable function $f'(x) = \omega(f(x))$ that can be decided using two pebbles. The main idea used in the proof is to simulate a Turing machine to decide $f(x) = y$ for some x and y by simulating a *program machine* [18] as described in the following section. $f'(x)$ is then chosen as a product of prime factors, where the exponent of one prime factor is $f(x)$, and the exponents of the additional factors are chosen to provide exactly the space required for the simulation. Throughout the algorithm's execution, the values of the program machine's registers are represented as the additional prime factor's exponents. If the Turing machine terminates with a positive result, and the given space exactly matches the required space for the simulation, the robot terminates with a positive result.

3.4 A Family of Functions Requiring an Increasing Amount of Pebbles

The discussion from the previous sections naturally brings up the question whether there is a function family whose detection requires an increasing amount of pebbles. In this section, we answer that question positively. As the proofs are fairly straightforward, we mostly state the general ideas, leaving out some details.

In order to simplify analysis, we consider a robot with pebbles operating on a line segment instead of a parallelogram. We also assume that pebbles are distinguishable and can be placed onto the same tile. Note that it is easy to simulate colors of a finite amount of pebbles on a line segment by keeping track of the pebbles' order. Furthermore, the robot can simulate placing two pebbles onto each other by placing the second pebble within constant distance instead and save the offset from its intended position.

First, we introduce the notion of *program machines* as defined in [18, Section 11].

▶ **Definition 7.** A *program machine* consists of a finite set of registers, holding arbitrary numbers from $\mathbb{N}_0$, and a finite program of numbered instructions from the following instruction set.

- $zero(r)$: Set register $r := 0$.
- $increment(r)$: Set register $r := r + 1$.
- $decrementOrJump(r, n)$: If $r = 0$, jump to instruction n. Otherwise set $r := r - 1$.
- $halt$: Stop the execution.

▶ Remark. Using the instructions from Definition 7, it is possible to simulate instructions to copy the value of register r_1 to r_2, and to jump if (or if not) $r_1 = r_2$ [18].

▶ **Lemma 8.** *A 3 register program machine can simulate a deterministic Turing machine with* $\Gamma = \Sigma = \{0, 1\}$.

Proof. It is well known that a Turing machine can be simulated using two stacks containing the tape's bits behind and in front of the head, respectively. These stacks can be viewed as the binary encoding of natural numbers. They will be stored in registers one and two.

Using the third register as a scratch pad, doubling and halving a register is simple. To pop a bit from the stack, we divide by two and look at the remainder. To push a bit onto the stack, we multiply by two and add the bit. For more details, we refer the reader to [18, Section 11]. ◀

Since we are interested in a robot moving on a finite space, we now define *bounded program machines*.

▶ **Definition 9.** A *bounded program machine* is a program machine whose first register is initialized to the input x. The other registers are initialized to 0. No register may exceed x.

The following observation follows from the fact that a robot can easily simulate a bounded program machine using pebbles, and vice versa.

▶ **Observation 10.** *The computational power of a robot with n pebbles on a line of length $x \geq n$ is between that of an n and an $n+1$ register bounded program machine with input x.*

The next two lemmas show that deterministic linear bounded automata are essentially equivalent in their computational capabilities to bounded program machines and that the number of registers relates to the number of tape symbols needed. This enables us to apply well-known results from complexity theory to our model. We use the definition of deterministic linear bounded automata from [7] as Turing machines that never leave the cells in which their input was placed. Inputs are restricted to $\{0,1\}^*$.

▶ **Lemma 11.** *A deterministic linear bounded automaton with $|\Gamma| = n$ and $\Sigma = \{0,1\}$ with input $x \in \Sigma^*$ can be simulated using a $1 + 2\lceil \log n \rceil$ register bounded program machine with input $x' := (1x)_2$.*

Proof. The construction from Lemma 8 ensures that no register will be increased beyond x' when simulating the linear bounded automaton. The two stacks will now contain elements in Γ. We encode the symbols in binary and store their representation using $2\lceil \log n \rceil$ registers. ◀

▶ **Lemma 12.** *An n register bounded program machine with input $(1x)_2, x \in \{0,1\}^*$ can be simulated by a deterministic linear bounded automaton with 2^n tape symbols and input x.*

Proof. The tape stores the binary representation of the registers' values, each symbol representing one bit of n registers. Operations from Definition 7 are now trivial to perform. ◀

Finally, it can be shown that there exists a family of languages requiring deterministic linear bounded automata with an increasing amount of tape symbols. These will directly translate to sets of accepted line lengths.

▶ **Lemma 13.** *There exists a family of context sensitive languages $S_n \subseteq \{0,1\}^*$ not accepted by any deterministic linear bounded automaton with fewer than n symbols [7, Corollary 2].*

Together with Observation 10, the previous lemmas imply the following theorem.

▶ **Theorem 14.** *There exists a family $L_n \subseteq \mathbb{N}$ such that a robot exists that can detect whether a line has length $\ell \in L_n$ using a finite number of pebbles but none using less than n pebbles.*

To construct a family of functions for deciding side ratios of a parallelogram, we can set

$$f_n(h) = \begin{cases} 1, & h \in L_n \\ 2, & h \notin L_n \end{cases}.$$

▶ Remark. The resulting parallelogram differs from the previous examples in that its length is only 1 or 2. Note that the robot can perform the simulation of the bounded program machine using the parallelograms height, and a length of 2 does not give it too much power. Furthermore, we can modify the function family such that $\ell \geq h$ while still requiring n pebbles to decide $f_n(h)$.

4 Future Work

In this paper we have identified some simple functions that can or cannot be decided with a given set of pebbles. Beyond our study in Section 3.4, we are interested to find functions, such as superexponentials, that require more than only two pebbles. Furthermore, it is an interesting question whether, instead or in addition to using pebbles, multiple robots can help in shape recognition problems. For example, it is still an open question whether two robots, which are activated in an arbitrary order, are more powerful than a single robot with a pebble. Apart from that, there are many different model assumptions under which our problems can be investigated.

In this work we primarily focused on detecting parallelograms of certain side ratios. As our ultimate goal is to investigate shape recognition in general, we are very interested to examine whether our results and algorithms are applicable to other shapes as well. For example, it may be possible to recognize more complex structures such as irregular hexagons with certain side ratios, or to even come up with a more generic procedure to recognize larger families of shapes. Furthermore, rasterized disks or ellipses might be interesting shapes to consider. Other intriguing problems related to shape recognition include testing symmetry or simply-connectedness of a tile structure.

References

1 M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978.

2 A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.

3 L. Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978.

4 S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.

5 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proc. 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.

6 Constantine Evans and Erik Winfree. DNA sticky end design and assignment for robust algorithmic self-assembly. In *Proc. of DNA Computing and Molecular Computing (DNA)*, pages 61–75, 2013.

7 Eliot D. Feldman and James C. Owings. A class of universal linear bounded automata. *Information Sciences*, 6(Supplement C):187–190, 1973. doi:10.1016/0020-0255(73)90036-4.

8 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, jun 2008.

9 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

10 R. Gmyr, I. Kostitsyna, F. Kuhn, C. Scheideler, and T. Strothmann. Forming tile shapes with a single robot. In *Abstr. European Workshop on Computational Geometry (EuroCG)*, pages 9–12, 2017.

11 F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Proc. International Fundamentals of Computation Theory Conference (FCT)*, pages 433–444, 1981.

12 S. Hong and Y. Yang. On the periodicity of an arithmetical function. *Comptes Rendus Mathematique*, 346(13):717–721, 2008.

13 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.

14 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Collectives of automata in labyrinths. *Discrete Mathematics and Applications*, 13(5):429–466, 2003.

15 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Independent systems of automata in labyrinths. *Discrete Mathematics and Applications*, 13(3):221–225, 2003.

16 E. Markou. Identifying hostile nodes in networks using mobile agents. *Bulletin of the European Association for Theoretical Computer Science*, 108:93–129, 2012.

17 Othon Michail and Paul G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.

18 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

19 Jitsuro Nagura. On the interval containing at least one prime number. *Proc. of the Japan Academy*, 28(4):177–181, 1952. doi:10.3792/pja/1195570997.

20 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

21 A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.

22 John H. Reif and Sudheer Sahu. Autonomous programmable DNA nanorobotic devices using dnazymes. *Theoretical Computer Science*, 410:1428–1439, 2009.

23 A. N. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3(3):236–246, 1974.

24 A.J. Thubagere, W. Li, R.F. Johnson, Z. Chen, S. Doroudi, Y.L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017.

25 S.F.J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A.J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.

26 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 *Proc. 4th Conference of Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.