Revisiting of 2-pebble automata from a dynamical approach

Camilo Lacalle* and Anahí Gajardo

Departamento de Ingeniería Matemática and Centro de Investigación en Ingeniería Matemática (CI²MA), Universidad de Concepción, Concepción, Chile

received , *revised* , *accepted* .

Pebble automata are at an intermediate point between finite automata, which can only read, and Turing machines: pebble automata cannot write but mark the tape with pebbles, of which they have a finite amount. On the square grid $\mathbb{Z}^2$, their ability to recognise figures and exiting labyrinths has been considered. Also, their ability to explore the *empty space* was studied. Their power depends strongly on the number of available pebbles. Here we put our attention on 2-pebble machines with 1 symbol, and we take the symbolic dynamics point of view by studying its *trace subshift*, *i.e.*, the set of sequences of 'states' and 'views' of the head during a complete execution. We prove that the trace subshift can be recognised with a 2-counter automaton in real time, but not, in general, with a 1-pushdown automaton.

Keywords: pebble automata, Turing machines, realtime recognition, symbolic dynamics

1 Introduction

Pebble automata were introduced by Blum and Hewitt (1967). A pebble automaton consists in a machine that moves over the grid, has an internal state, can read the symbols over the grid, and has a set of pebbles that it can use to mark the cells. In their seminal paper, they defined a hierarchy in the sets of bounded figures, determined by the minimal quantity of pebbles needed by the machine to recognise them. There, the machine moves on the finite rectangle where the figure is defined. In this context, they prove that the hierarchy is strict and infinite.

Delorme and Mazoyer (1999) continued this work by considering pebble machines over the whole grid $\mathbb{Z}^2$. There, the hierarchy collapses, because a 3-pebble machine results to be Turing universal; then, it can recognise all the recognisable families but, in order to do this, it needs to localise its pebbles at arbitrarily large positions. They also considered machines with only one symbol. Their interest in considering this

*This work has been supported by CONICYT FONDECYT #1090568 and BASAL project CMM, Universidad de Chile, and CI²MA, Universidad de Concepción.
Email:clacalle@udec.cl

particular case is to determine its capability to "explore the world". If we want a machine to be able to explore the world, it must be able to do it independently on the pattern depicted on it. It must do it even without taking into account this pattern, i.e., as if there were only one symbol on the grid. A 3-pebble machine with one symbol is also Turing universal, thus it is also capable of exploring the world. 2-pebble machines are strictly less powerful; Delorme and Mazoyer prove that these machines can only cover a conical part of the grid, and that they eventually have a very regular behaviour. 1 and 0-pebble machines are even less complex: they have an eventually periodic movement.

Pebble and Turing machines are particular cases of one-head machines. These machines have been recently studied from the point of view of symbolic dynamics (Gajardo and Mazoyer (2007); Gajardo (2008); Gajardo and Guillon (2010)). The idea consists in recording, for each initial configuration, the infinite sequence of read symbols and internal states that the machine has through time. Such sequence contains all the relevant information to recover the head trajectory and the part of the initial configuration that has been visited by the machine. The set of these sequences is a subshift that we call t-shift.

The complexity of the t-shift of a machine reflects the complexity of the machine itself in a particular way. Gajardo and Mazoyer (2007) remark that the t-shift of any one-dimensional machine can be recognised in real time by a two-pushdown automaton, while the t-shift of some two-dimensional finite automata cannot be recognised with a deterministic pushdown automaton with any number of pushdown stores.

Machines with 0 and 1-pebble were already studied in this context in Gajardo (2008), where it is proved that the associated t-shift is sofic, *i. e.*, it can be recognised by a finite automaton. 2-pebble automata can cover a two-dimensional part of $\mathbb{Z}^2$, but they do it by zigzagging through the grid, as it is shown in Delorme and Mazoyer (1999). This suggests the main result of this paper: the t-shift of a 2-pebble automata can always be recognised with a two-pushdown automaton in real time, more precisely with a two-*counter* automaton.

When studying the complexity of t-shifts, we consider the hierarchy of *real time* recognisability. This notion is defined in the context of formal languages (see, for example, Wagner and Wechsung (1986)). The hierarchy defined by deterministic k-pushdown automata is infinite as shown in Aanderaa (1974). The same is valid for the hierarchy defined by deterministic k-counter automata in Fischer et al. (1968). Real time recognisability of languages extends in a natural to recognisability of subshifts: the automaton receives exactly one symbol at each iteration. If at some point a word outside the language is received, the automaton will reject it immediately, while a correct infinite word will induce an infinite computation. Real time subshifts are studied by Kůrka and Maass in Kůrka and Maass (2000), where the hierarchy is proved to be a topological invariant, and several key examples are developed.

In the next two sections, we give formal definitions and recall some useful results of Delorme and Mazoyer (1999). In section 4, it is proved that a one-pushdown automaton is, in general, not enough (section 4) to recognise the t-shift of a 2-pebble automaton with 1 symbol. Section 5 establishes some elementary facts about the geometry in $\mathbb{Z}^2$. The general 2-counter automaton that recognises an arbitrary 2-pebble machine is constructed in section 6.

2 Formal definitions

2.1 The pebble machine

We consider a pebble machine as the tuple $M = (Q,\ \delta,\ \lambda,\ k)$, where Q is the finite set of inner states, δ and λ are the state transition function and the site transition function respectively, and k is the amount of

available pebbles. We define the transition functions as follows.

$$\begin{array}{rccc} \lambda: & Q\times\{0,1\}^k\times\{0,1\}^k & \longrightarrow & \mathcal{D}\times\{0,1\}^k \\ & (q,\nu,\eta) & \longrightarrow & \lambda(q,\nu,\eta)=(\hat{d},\eta^\star) \end{array}$$

$$\begin{array}{rccc} \delta: & Q\times\{0,1\}^k\times\{0,1\}^k & \longrightarrow & Q\times\{0,1\}^k \\ & (q,\nu,\eta) & \longrightarrow & \delta(q,\nu,\eta)=(q^\star,\nu^\star) \end{array}$$

Here $\mathcal{D}=\{(0,1),(1,0),(0,-1),(-1,0)\}$. The vector ν represents the pebble currently charged by the machine, while η represents the pebble on the ground under the machine's head. The number of pebbles is conserved and there is only one copy of each pebble, thus $\nu+\eta=\nu^\star+\eta^\star\leq 1$.

2.2 Formal languages

Given a finite set A, the set A^* stands for the whole set of finite sequences of elements of A, called *words*, including the empty sequence denoted by e. Given two words $u=u_1...u_n$ and $v=v_1...v_m$, their *concatenation* is the word $uv=u_1...u_nv_1...v_m$. In the same way, u^i denotes u concatenated i times with itself. A word u is a *factor* of v ($u\sqsubseteq v$) if there exist two words x,y such that $xuy=v$, and u is a *prefix* of v ($u\preceq v$) if $x=e$. We say that two words u, v *overlap* if there exist words x, w, y such that $u=xw$ and $v=wy$. A *language* in A is a subset $\mathcal{L}\subseteq A^*$.

Following Kůrka and Maass (2000), we will consider the notion of *realtime recognition by k-pushdown automata*. There it is said that a language is in $\mathbb{Q}(k)$ if it is recognized by a k-pushdown automaton in realtime, and that it is in $\mathbb{R}(k)$ if it is recognized by a deterministic k-pushdown automaton in realtime. $\mathbb{R}(0)=\mathbb{Q}(0)$ is the set of regular languages, recognized by finite automata. $\mathbb{Q}(1)$ is the set of context free languages, recognized by 1-pushdown automata. $(\mathbb{R}(k))_{k\in\mathbb{N}}$ defines an infinite hierarchy while $(\mathbb{Q}(k))_{k\in\mathbb{N}}=(\mathbb{Q}(k))_{k=0}^3$.

A k-counter automaton is an automaton with a finite memory that can, additionally, store k integers, being able to increment, decrement and test whether they are zero or not. Some authors consider that counters can only contain natural numbers, but the sign can be easily implemented through the state set. We will consider k-counter automata that can also *reset to 0* some of their counters. This automata can be implemented with a k-pushdown automaton having a pushdown alphabet with two symbols. Thus, languages recognized in realtime by a deterministic k-counter automaton with *reset* instruction are in $\mathbb{R}(k)$. We give now precise definitions.

A deterministic k-counter automaton with integer counters and reset instruction is a structure $M=(A,\Omega,k,\varphi,o_0,F)$, where A is the input alphabet, Ω is the states set, k is the number of counters, $\varphi: A\times\Omega\times\{0,*\}^k\to\Omega\times\{-1,0,1,\textit{reset}\}^k$ is the transition function, $o_0\in\Omega$ is the initial state and $F\subset\Omega$ is the set of final states.

A configuration of the system is a tuple $(q,n_1,..,n_k)\in\Omega\times\mathbb{Z}^k$. The automaton can only *distinguish* if the counters are 0 or not, *i. e.*, it only sees the tuple $(q,s(n_1),..,s(n_k))$, where $s:\mathbb{Z}\to\{0,*\}$, assigns a 0 to 0 and * to other numbers. Four actions are defined α_{-1}, α_0, α_1 and $\alpha_{\textit{reset}}:\mathbb{Z}\to\mathbb{Z}$; the first three consist in adding -1, 0 or 1, and the last one resets the counter to 0.

Thus, if the current symbol is a and $\varphi(a,q,s(n_1),..,s(n_k))=(q',a_1,..,a_k)$, then $(q,n_1,..,n_k)$ evolves to $(q',\alpha_{a_1}(n_1),..,\alpha_{a_k}(n_k))$.

The *language L_M accepted in realtime* by M consists of all words w in A^* such that $(q_0,0,..,0)$ evolves to a configuration with an accepting state (in F) by successively reading each symbol of w. We

remark that exactly one symbol must be *read* at each time step; this is the reason to say that this is *realtime* recognition.

2.3 Subshifts

Given a finite set A, the set $A^{\mathbb{N}}$ stands for the one sided full-shift. It contains infinite sequences of symbols in A, or infinite words. Concepts defined for words can be analogously defined for infinite words. Thus we can concatenate a finite word with an infinite word, and we will say that $u \in A^*$ is a factor of $z \in A^{\mathbb{N}}$ if there exist two words $x \in A^*$ and $y \in A^{\mathbb{N}}$ such that $xuy = z$. The shift function $\sigma : A^{\mathbb{N}} \to A^{\mathbb{N}}$ is defined by $(\sigma(z))_i = z_{i+1}$, for every $i \in \mathbb{N}$. Given a language $L \subseteq A^*$, a set of infinite words can be defined by

$$\mathcal{S}(L) = \{z \in A^{\mathbb{N}} \mid (\forall u \sqsubseteq z)\ u \in L\}.$$

Also, given a set $\Sigma \subseteq A^{\mathbb{N}}$, its associated language is

$$\mathcal{L}(\Sigma) = \{u \in A^* \mid (\exists z \in \Sigma) u \sqsubseteq z\}.$$

In general $\mathcal{S}(\mathcal{L}(\Sigma)) \supseteq \Sigma$ and $\mathcal{L}(\mathcal{S}(L)) \subseteq L$. If Σ satisfies $\mathcal{S}(\mathcal{L}(\Sigma)) = \Sigma$, then it is called *subshift*.

We say that a subshift Σ is recognized by a k-counter automaton in realtime if its associated language has this property.

2.4 Pebble machines and their associated trace

Given a k-pebble machine $M = (Q,\ \delta,\ \lambda,\ k)$, its associated dynamical system is (X, M), where $X = \left\{(s, q, \nu, g) \in (\{0,1\}^k)^{\mathbb{Z}^2} \times Q \times \{0,1\}^k \times \mathbb{Z}^2 \mid (\forall i \in \{1,..,k\})\ \#\{p | s(p)_i = 1\} + \nu_i = 1\right\}$ and M is overloaded to also denote the transition function of the automaton.

Let $\pi : X \to Q \times \{0,1\}^k \times \{0,1\}^k$ be defined by $\pi(s, q, \nu, g) = (q, \nu, s(g))$ and let $\psi : X \to (S \times Q)^{\mathbb{N}}$ be defined by $\psi(x) = (\pi(M^n(x)))_{n\in\mathbb{N}}$. The *t-shift* of (X, M) is $S_M = \psi(X)$.

The set $\psi(X)$ represents all the possible sequences of triplets (state, charge, view) that the machine can produce when considering all the possible initial configurations. The associated language by L_M is the set of finite subsequences of S_M.

3 Previous results

Delorme and Mazoyer (1999) studied k-pebble machines. Their first result establishes that a 1-pebble machine has an eventually-periodic behavior, and it gives bounds for the transient, period and the geometry of this movement. We recall some of their results in the next lemma, by giving a new proof.

Lemma 3.1 (Delorme and Mazoyer (1999)) *A one pebble machine with n states that start on a state q always falls in an eventually periodic movement of transient time t_p and period p_q such that $t_q + p_q \leq n^3$, and only one of the next four possibilities can happen.*

1. *The machine ignores the pebble and follows a preperiodic movement such that $t_q + p_q \leq n$.*
2. *The machine leaves the pebble and follows a periodic movement with $p_q \leq n$.*

3. *After a transient time, the machine follows a periodic movement with $p_q \leq n$, where it permanently carries the pebble.*

4. *The machine follows a preperiodic movement where it drops and picks the pebble an infinite number of times and $p_q + t_q \leq n^3$.*

Proof:

1 If it ignores the pebble, it behaves as a 0-pebble machine. In this case, the sequence of states is completely determined by the initial state q. Thus, when some state is repeated, the machine enters into a periodic phase. By the pigeon hole lemma, this happens at time $t_q + p_q \leq n$.

2,4,3 If the machine uses its pebble, it will alternately carry and drop the pebble. If some state is repeated while the machine carries the pebble, a periodic phase begins. Thus, the number of time steps with the pebble before this phase is bounded by n. In between, the machine can drop the pebble many times, but always coming back to the position where it was dropped. The machine can stay more than n steps in these rounds without the pebble, but in this case the movement will have a sort of local transient of length say i and a sort of local periodic phase of length j that it repeats l times before founding the pebble again. Clearly $i + j \leq n$, but we need a bound for l. In the transient phase the head can go up to a distance at most i from the pebble. During the *sub*periodic phase it will move at least one unit each time, thus it cannot repeat this more than i times. Thus the total time is $i + ij \leq n^2$. Now, counting both the phases with and without the pebble, we conclude that the global transient plus the period is less or equal than n^3. If during the periodic part the machine does not drop the pebble or it does not carry it, the period is bounded by n.

□

From this result, it is not difficult to see that the t-shift associated to a 1-pebble machine can be recognized by a finite automaton, i.e., it is *sofic*. This is developed with detail in Gajardo (2008).

2-pebble machines are more interesting, in the sense that they can present a non-eventually periodic movement, because the head can oscillate between its two pebbles by putting them each time farther away. An example of this is developed in section 4, where the machine is not following a periodic movement but a very regular one. Delorme and Mazoyer showed that any 2-pebbles machine either falls into a behaviour like this or into a periodic one.

4 One stack is not enough

In this section we use the pumping lemma for Context-Free Languages (CFL) to prove that the t-shift of a 2-pebble machine is not a CFL in general. For this purpose, we provide a particular 2-pebble machine whose t-shift fails to satisfy the pumping lemma.

Let $M = (Q, \Sigma, \delta, \lambda, 2)$ be a 2-pebble machine with $Q = \{q_0, q_1, q_2, q_l, q_r\}$, and δ and λ given by table 1. For simplicity, we have ignored the transitions which are not needed for our goal.

The alphabet of the t-shift is the set $Q \times \{0,1\}^2 \times \{0,1\}^2$. The table defines also an abbreviation for each pertinent tuple in this alphabet, which will be useful in the analysis of the t-shift.

Figure 1 exhibits the behavior of M starting with the configuration abbreviated by 0. The whole evolution of the machine is specified by this initial configuration, where the head holds both pebbles. This

$(q,\ (\nu_1,\nu_2),\ (\eta_1,\eta_2))$	Abbr.	$\delta(q,\ (\nu_1,\nu_2),\ (\eta_1,\eta_2))$	$\lambda(q,\ (\nu_1,\nu_2),\ (\eta_1,\eta_2))$
$(q_0,\ (1,1),\ (0,0))$	0	$(q_2,\ (0,1))$	$((1,0),\widehat{i})$
$(q_0,\ (0,0),\ (0,0))$	d	$(q_r,\ (0,0))$	$((0,0),\widehat{i})$
$(q_0,\ (1,0),\ (0,0))$	a	$(q_2,\ (1,0))$	$((0,0),-\widehat{i})$
$(q_2,\ (0,0),\ (0,0))$	c	$(q_0,\ (0,0))$	$((0,0),\widehat{j})$
$(q_0,\ (0,1),\ (0,0))$	e	$(q_2,\ (0,1))$	$((0,0),\widehat{i})$
$(q_2,\ (0,1),\ (0,0))$	f	$(q_l,\ (0,0))$	$((0,1),-\widehat{i})$
$(q_2,\ (1,0),\ (0,0))$	b	$(q_2,\ (0,0))$	$((1,0),\widehat{i})$
$(q_r,\ (0,0),\ (0,0))$	r	$(q_r,\ (0,0))$	$((0,0),\widehat{i})$
$(q_r,\ (0,0),\ (0,1))$	$\overline{r}$	$(q_0,\ (0,1))$	$((0,0),-\widehat{j})$
$(q_l,\ (0,0),\ (0,0))$	l	$(q_l,\ (0,0))$	$((0,0),-\widehat{i})$
$(q_l,\ (0,0),\ (1,0))$	$\overline{l}$	$(q_0,\ (1,0))$	$((0,0),-\widehat{j})$

Tab. 1: Local transition rules δ and λ for M and the abbreviation of the corresponding local configurations.

implies that only one word is possible after the symbol '0'. The next claim characterizes this word and states some of its properties, which will be useful for proving our result.

Claim 4.1 *Given a word $w \in (Q \times \{0,1\}^2 \times \{0,1\}^2)^*$ such that $w_0 = 0$, it holds that:*

- *$w \in L_M$ if and only if w is a prefix of $S = 0f\bar{l}\prod_{i=1}^{\infty} s_i$; where $s_i = abcdr^{2(i-1)}\overline{r}efl^{2i}\bar{l}$.*
- *If w is a prefix of S ending with $fl^{2n}\bar{l}$ then $w = 0f\bar{l}\prod_{i=1}^{n} s_i$.*

Proposition 4.1 *L_M is not a CFL.*

Proof: Suppose L_M is a CFL. By the pumping lemma for CFL (see Hopcroft et al. (2001), for example), there exists $n \in \mathbb{N}$ such that for all $z \in L_M$ with $|z| \geq n$, there exist u, v, w, x, y satisfying:

1. $z = uxvyw$
2. $|xvy| \leq n$
3. $xy \neq \varepsilon$
4. $\forall i \geq 0,\ ux^ivy^iw \in L_M$.

Let us choose $z = 0f\bar{l}\prod_{i=1}^{n} s_i$. Then we have 3 cases depending on how $uxvyw$ partition z. We remark first that the length of s_n is $4n + 6$.

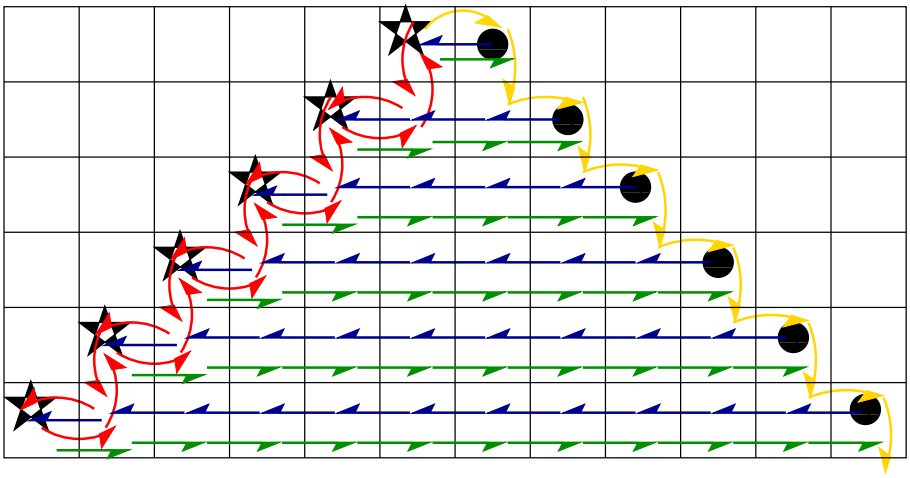

Fig. 1: Simulation of M when it starts in state q_0 and charging both pebbles , i. e., in situation '0'. The head oscillates between the pebbles by moving them 1 case away each time. One pebble is represented by a star and the other by a circle.

A. $|w| \geq 2 + 2n$.

In this case, ux^2vy^2w ends with $fl^{2n}\bar{l}$. Thus, by claim 4.1 and 4 $uxvyw = ux^2vy^2w = z$, which is a contradiction with 3.

B. $|w| = 0$.

By 2, $xvy = l^k\bar{l}$, for some $k < n$. Thus ux^2vy^2w starts with 0 but it is not a prefix of S.

C. $0 < |w| < 2 + 2n$.

In this case the word xvy is fully contained in s_n. We can say that $s_n = aAxvyB\bar{l}$, where A and B are factors of s_n that do not contain a nor $\bar{l}$. Since a and $\bar{l}$ are delimiters of all the s_k, the word $aAx^2vy^2B\bar{l}$ may contain only one s_k, and since $ux^2vy^2w = 0f\bar{l}\left(\prod_{i=1}^{n-1} s_i\right) aAx^2vy^2B\bar{l}$, we conclude that $ux^2vy^2w = uxvyw$, which is again a contradiction.

□

5 Geometrical facts

The behaviour of Pebble Machines is highly influenced by the geometrical characteristics of the Euclidean space $\mathbb{Z}^2$. In this section we present some intuitive geometrical remarks that will be intensively used in section 6. We start by giving some definitions.

Definition 5.1 *In $\mathbb{R}^2$ the norm $||(x_1, x_2)||_p = (|x_1|^p + |x_2|^p)^{\frac{1}{p}}$ is well known, particularly in $\mathbb{Z}^2$ we consider the Hamming norm: $||(x_1, x_2)||_1 = |x_1| + |x_2|$, and the Hamming distance $d(x, y) = ||x - y||_1$.*

A ball of radius r and center x is thus the set $B(x,r) = \{y \in \mathbb{Z}^2 \mid d(x,y) \leq r\}$ and its border the set $Fr(x,r) = \{y \in \mathbb{Z}^2 \mid d(x,y) = r\}$. Given a number n, and a set $A \subseteq \mathbb{Z}^2$, we consider $nA = \{na \mid a \in A\}$. Given a vector v we consider $v + A = \{v + x \mid x \in A\}$ and given a set $R \subseteq \mathbb{R}$, we consider $Rv = \{rv \mid r \in R\}$. Given two vectors $x, y \in \mathbb{R}^2$, the interval $[x,y] = \{tx+(1-t)y \mid t \in [0,1]\}$ denotes the convex hull of $\{x,y\}$.

Lemma 5.1 *Given a convex set $B \subseteq \mathbb{R}^2$, a natural number m and three vectors u, v, $w \in \mathbb{Z}^2$, if the set $R = w + \mathbb{R}_+ u$ satisfies that $B \cap R$ and $(B + mv) \cap R$ are non empty, then for every $k \in \{0, .., m\}$ $(B + kv) \cap R \neq \emptyset$.*

Proof: Let $y_0 \in B \cap R$, and $y_m \in (B + mv) \cap R$, then there exist r_0, $r_m \in \mathbb{R}$ and b_0, $b_m \in B$ such that $y_0 = r_0 u + w = b_0$ and $y_m = r_m u + w = b_m + mv$.

By convexity, $[b_0, b_m] \subseteq B$. Thus, for any $k \in \{0, 1, 2, \ldots, m\}$, $b_k = \frac{m-k}{m} b_0 + \frac{k}{m} b_m \in B$ and $y_k = b_k + kv \in B + kv$. Finally,

$$\begin{aligned} y_k &= \frac{m-k}{m}(r_0 u + w) + \frac{k}{m}(r_m u + w - mv) + kv, \\ &= \frac{r_0(m-k) + k r_m}{m} u + w, \\ &\in R \cap (B + kv). \end{aligned}$$

□

Lemma 5.2 *Given a finite set $S \in \mathbb{Z}^2$, a vector $u \in \mathbb{Z}^2$ of norm $||u||_1 \leq r$ and a point $x \in \mathbb{Z}^2$ at a distance greater than $l > 0$ from the set $S + \mathbb{N}u$, then x is at a distance greater than $l - r$ from the set $S + \mathbb{R}_+ u$.*

Proof: Let $z \in S + \mathbb{R}_+ u$, then there exist $y \in S + \mathbb{N}u$ and $\lambda \in [0,1]$ such that $z = y + \lambda u$. In this way $d(z,y) = \lambda ||u||_1 \leq r$. Then $l < d(x,y) \leq d(x,z) + d(z,y) \leq d(x,z) + r$, and thus $l - r < d(x,z)$. Since z is an arbitrary element of $S + \mathbb{R}_+ u$, the result is concluded. □

Lemma 5.3 *Given a finite set $S \subseteq \mathbb{Z}^2$, three vectors u, v, $w \in \mathbb{Z}^2$ and $r \in \mathbb{N}$ such that $S \subseteq B(w,r)$ and $||u||_1$, $||v||_1 \leq r$, then, if $m \in \mathbb{N}$ satisfies that $B(w + mv, 2r) \cap (S + \mathbb{N}u) = \emptyset$, then for every $n \geq m$, $B(w + nv, r) \cap (S + \mathbb{N}u) = \emptyset$.*

Proof: Let us define $B = B(w,r)$, then $S + \mathbb{N}u \subseteq B + \mathbb{N}u$. Let us suppose that there exist $y \in S$, $k \in \mathbb{N}$ and $n > m$ in $\mathbb{N}$ such that $y + ku \in B + nv$. Let us define $R = y + \mathbb{R}_+ u$. We have that $y \in B \cap R$ and $y + ku \in (B + nv) \cap R$, thus, by Lemma 5.1 $(B + mv) \cap R \neq \emptyset$.

Now, by taking $x = w + mv$ and $l = 2r$ we have that x is at a distance less than r from $S + \mathbb{R}_+ u$. We can use the counterproposal of Lemma 5.2 to state that $B(x, 2r) \cap (S + \mathbb{N}u) = B(w + mv, 2r) \cap (S + \mathbb{N}u) \neq \emptyset$, which is a contradiction. □

6 Recognisability with a 2-counter automaton

In this section we construct a 2-counter automaton which recognizes the t-shift of a given arbitrary 2-pebble machine $M = (Q,\ \delta,\ \lambda,\ 2)$ in realtime. The automaton works by storing what the machine should have seen to produce the input word, as we describe here. First, let us observe that from the input word we can deduce the trajectory that the machine has followed, *i.e.*, we can deduce which cells have been visited and where the pebbles have been found. In order to check the correctness of a word, we need to verify the following points:

1. If a cell has been visited and it was empty, then at the next visit it must be also empty.
2. If a cell has been visited and the machine has left a pebble, then at the next visit the pebble must be there.
3. The sequence of inner states and pebble load must be coherent.

To verify these points, the automaton should store the following information:

1. The current state and pebble load.
2. The whole set of visited cells until the second pebble is found (since if both pebbles have been found, the set of visited cells is not needed anymore).
3. The current relative position of the pebbles that have already been found.

The whole set of visited cells and the relative position of the pebbles can be arbitrary large. Then, they will be codified through different techniques depending on the current situation. We distinguish five stages in time, which decompose the state set of the automaton into five sets as follows.

$$\Omega = \Omega_A \cup \Omega_B \cup \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \{\theta,\ R\}$$

Where θ is the initial state and R is the reject state. We define $n = |Q|$ and $r = n^3$, which is an upper bound for the transient and the period of the machine when it has only one pebble. Each set is described in Table 2.

Stage i	State Ω_i
1	$([1], [(\overline{q}, \overline{\nu}, \overline{\eta}), (V_0, \emptyset, \emptyset), (d_0, d_T, 0)], [(u, 0, T), 0])$
A	$([A], [(\overline{q}, \overline{\nu}, \overline{\eta}), (V_0, \emptyset, U), (d_0, 0, d_p)], [(u, v, T), \rho])$
2	$([2], [(\overline{q}, \overline{\nu}, \overline{\eta}), (V_0, V, \emptyset), (0, d_T, 0)], [(u, 0, T), 0])$
B	$([B], [(\overline{q}, \overline{\nu}, \overline{\eta}), (V_0, V, U), (0, d_T, d_p)], [(u, v, T), \rho])$
3	$([3], [(\overline{q}, \overline{\nu}, \overline{\eta}), (d_1, d_2), (\sigma_1, \sigma_2)])$

Tab. 2: This table presents the states associated to each stage. The coordinates equal to 0 or $\emptyset$ are fixed for the whole stage. The first component is the stage, the second component contains the variables that change on every step and, for the first four stages, the third component contains parameters which are fixed once for the whole dynamics.

As table 2 specifies, the first part of the state is $(\overline{q}, \overline{\nu}, \overline{\eta})$, which is precisely the state that the pebble machine should have, its pebble load and the pebble content at the head position, respectively. If $(\overline{q}, \overline{\nu}, \overline{\eta})$

is not compatible with the input symbol, the automaton changes to state R, and it rejects the input word. The variables $(\overline{q}, \overline{\nu})$ are updated at each iteration, following the transition rule λ of the machine. The pebble load is maybe unknown, thus the variable $\overline{\eta}$ can take values in $\{0,1\}^2 \cup \{\Phi\}$, where Φ symbolizes that the current cell has not been visited yet and it may contain pebbles or not. The variable $\overline{\eta}$ can be computed from the other variables that contain information about the visited cells and pebble positions. We explain now why we need to do this separation and how these stages are related.

6.1 Stages description

- Stage 1) *no pebble found and not far from the starting cell.* This is the first stage, where no pebble has been found yet. After a transient time of at most n iterations, the machine will fall into a periodic movement with a drift vector u of size at most n. The automaton will record the set of visited cells in a *fixed window* of radius $6r$ centered on the origin, called V_0. The position of the head is recorded in vector d_0. The particular cell $T \in V_0$ where the machine attains a distance of $3r$ from the origin is recorded as a vector too, to deserve as an alert point when the machine eventually cames back. At this point, the machine follows a periodical movement with drift vector u that can hold for long time. The relative position to T will be recorded as $d_T + Nu$, where N is the content of the counters (both) and d_T is a vector in $B(0, r)$. This stage finishes when the machine exits this window or when it finds a pebble.

- Stage 2) *the machine is far from the starting point and still no pebble is found.* The set of visited cells, which is periodic, will be recorded in a *moving window* V centered on the head position and with radius $4r$. This stage finishes only if one pebble is found.

- Stage B) *one pebble has been found far from the origin.* This stage is attained when a pebble, say ρ, is found during stage 2. This may produce a new periodic movement, with drift vector v. If the pebble is left behind, its relative position will also be recorded with counter ρ and vector d_p as $Kv + d_p$, where K is the number on the counter. The cells visited during this stage will be recorded in a moving window U of radius $4r$ centered on the head position. The cells visited during stage 2 are still maintained in window V, from which we may deduce this periodic set through a method called *back recovering*. The relative position to cell T is still recorded in one of the counters, in case the head comes back to the origin.

- Stage A) *one pebble has been found and the machine is near the origin again.* We are in this stage if either the pebble has been found during stage 1 or the machine has come back to V_0 after finding the pebble outside. We detect this last situation because the counter recording the position of cell T is empty. The cells visited before finding the pebble are recorded in the fixed window V_0, and the cells visited after finding the pebble will be recorded in the moving window U, just like in stage B. The relative position of the pebble is still recorded in the counter. It is possible that the machine exits V_0 again. In such case, the information contained in V_0 is simply ignored, because at this point only the cells visited after the finding of the pebble can be revisited before finding the second pebble.

- Stage 3) *both pebbles found.* At this point, just knowing the relative positions of each pebble is enough to test coherence, thus the automaton "forgets" all the windows. Vector σ_i records the drift vector of the periodic movement corresponding to the state that it had the last time it left pebble i

by going away with no pebbles in charge (if this does not happens $\sigma_i = 0$). Thus, the position of the pebbles can be recorded with the counters and vectors d_1 and d_2, as $N_1\sigma_1 + d_1$ and $N_2\sigma_2 + d_2$ as we have explained before.

Transitions between these sets are described in figure 2. Figure 3 shows the trajectory of the head and the contents of the different variables in Stage 2.

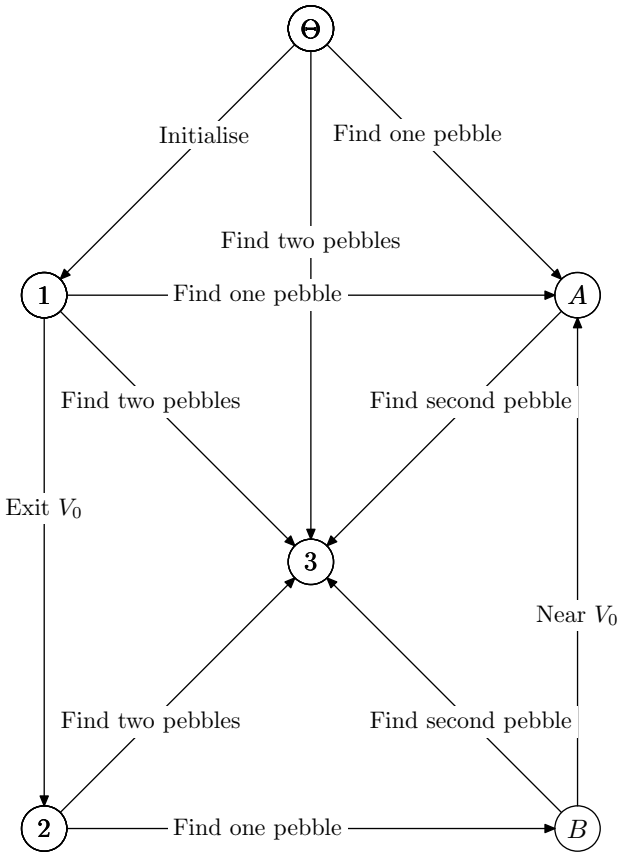

Fig. 2: This graph represents the stage transitions. Each node on the graph is a different stage, which depends on the number of found pebbles, and the distance from the starting point. The labels on the arrows are the conditions for the stage transition.

Now we describe the updating method for each of the variables. We have already described $(\overline{q}, \overline{\nu}, \overline{\eta})$. The vector d_0 represents the relative position of the head to the origin. It is updated by simply adding the current head movement, given by the transition function δ evaluated on the current state. There are four other vectors d_T, d_p, d_1 and d_2; they are used in combination with the counters in a way that we will explain in subsection 6.3. There are three windows (V_0, V, U); V_0 is a subset of $B(0, 6r)$ and the other two are subsets of $B(0, 4r)$. Each window is updated in a different way. To V_0 we simply add the cell

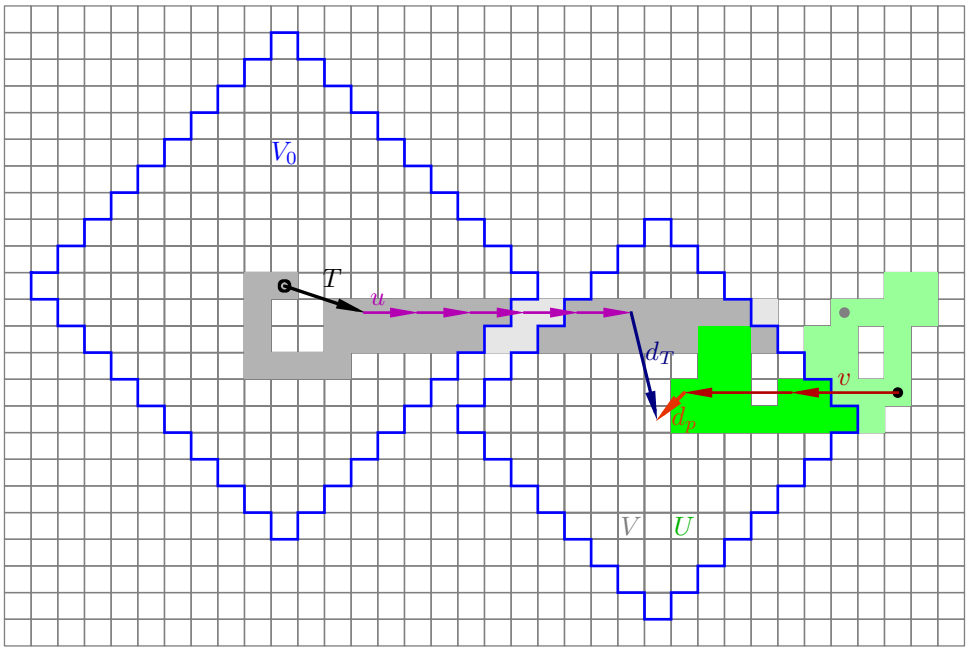

Fig. 3: A typical situation in Stage 2. The starting point is denoted by ∘ . The gray dot denotes the position where the pebble ρ was found by the first time. The black dot denotes the current position of the pebble. Colored squares are the visited cells, darker cells are those recorded with the windows. Gray cells are visited before founding the pebble, while green cells are visited afterward. In this example, counter $\neg\rho$ is contains a 5 and counter ρ contains a 2. The scale is not respected, windows should be larger.

corresponding to the current position d_0. U and V are updated either with the method of *moving window* or with the one of *moving window with back recover* depending on the current stage. These methods need to be defined in a precise way. We will formally prove their effectiveness in subsection 6.2. Variables (u, v, T, ρ) are updated only once, they are constants. Vectors u, v, σ_1 and σ_2 are the drift vectors of the periodic phases of the machine. They can be obtained thanks to lemma 3.1 which says that when no pebble is present, the machine will soon fall into a periodic movement which depends only on the initial state q, let us call such vector v_q. Vector u can be computed when the machine starts, v is computed when the first pebble is found, it will be equal to 0 if the periodic phase includes the pebble, and is it defined as v_q if q is one of the states of the periodic phase. Vectors σ_1 and σ_2 are initialized as 0, and σ_i is updated as v_q when the machine drops pebble i with state q and has no pebble load, while this does not arrive, σ_i keeps its ancient value. Of course, σ_i will change if the machine crosses pebble i again. T is equal to d_0 when the machine attains the frontier of the ball $B(0, 3r)$ in stage 1. The variable ρ represents the index of the first pebble found, if there is any. Table 3 specifies the way the variables are updated from one stage to the next.

Arc	Formal condition	Change of state
$(\theta, 1)$	$\nu = 0$ $\eta = (0,0)$	$G = 1$; $u = v_q$ $V_0 = \emptyset$; $d_0 = 0$;
(θ, A)	$\nu + \eta = (1,0)$ or $(0,1)$ $\eta = 0$	$\rho = \begin{cases} 1 & \text{if } \eta = (1,0) \\ 2 & \text{if } \eta = (0,1) \end{cases}$ $V_0 = U = \emptyset$; $d_0 = 0$; $d_p = 0$ $G = A$; $u = 0$; $v = 0$;
$(1, A)$	$\eta \neq 0$	$\rho = \begin{cases} 1 & \text{if } \eta = (1,0) \\ 2 & \text{if } \eta = (0,1) \end{cases}$ $U = \emptyset$; $d_p = 0$; $\mathit{counter}_\rho$=0 $G = A$; $v = 0$;
$(1, 2)$	$\|\|d_0\|\|_1 = 6r$	$G = 2$; $V = (V_0 - d_0) \cap B(0, 4r)$
$(2, B)$	$\eta \neq 0$	$\rho = \begin{cases} 1 & \text{if } \eta = (1,0) \\ 2 & \text{if } \eta = (0,1) \end{cases}$ $U = \emptyset$; $d_p = 0$ $G = B$; $v = 0$;
(B, A)	$\mathit{counter}_{\neg\rho} = 0$	$G = A$; $d_0 = d_T + T$
$(A, 3)$ $(B, 3)$	$\eta_{\neg\rho} = 1$	$G = 3$; $d_\rho = d_p$; $d_{\neg\rho} = 0$ $\sigma_{\neg\rho} = 0$; $\sigma_\rho = v$ $\mathit{counter}_{\neg\rho} = 0$
$(1, 3)$ $(2, 3)$	$\eta = (1,1)$	$G = 3$; $d_1 = d_2 = 0$; $\sigma_i = 0$ $\mathit{counter}_1 = \mathit{counter}_2 = 0$
$(\theta, 3)$	$\nu + \eta = (1,1)$	$G = 3$; $d_1 = d_2 = 0$; $\sigma_i = 0$ $\mathit{counter}_1 = \mathit{counter}_2 = 0$

Tab. 3: This table presents what we call "the stage transitions". They are performed when changing from one stage to the next, after the "usual" state transitions of each stage.

6.2 Updating the windows

Definition 6.1 *A* moving window *is a set $W \subset B(0,c)$ that is shifted at each iteration in the direction of the movement of the machine $\hat{d} \in \mathcal{D}$. This is performed by the following function.*

$$\begin{array}{rccl} H_c^{\hat{d}}: & \mathcal{P}(B(0,c)) & \longrightarrow & \mathcal{P}(B(0,c)) \\ & W & \longrightarrow & H_c^{\hat{d}}(W) = [(W \cup \{0\}) - \hat{d}] \cap B(0,c) \end{array}$$

Next lemma assures that the information contained in the moving window is correct, at least at the center of the window.

Lemma 6.1 *If the machine is following a periodic movement of period $p < r$, and the window W is being updated through the function $H_c^{\hat{d}}$, where $\hat{d}$ corresponds to the current head movement and $c > r$, then the set $W \cap B(0, c-r)$ contains exactly all the relative positions of the cells that have been visited in a radius $c - r$ of the head.*

Proof: We suppose that the window is correctly initialised with the set of visited cells in a radius c.

At each iteration the cell 0 is added, which is correct since the head is at cell 0. At the same time, the head has moved in direction $\hat{d}$ and consequently the cells in W have been shifted in this direction. Thus all the cells in W have been already visited.

On the other hand, the only way a cell can exit W is by attaining a distance c from the origin. This means that it has been moved from the origin up to a distance $c > p$. This cell cannot enter the ball $B(0, c-r)$ during a single period, and after that it will be even farther. □

Definition 6.2 *A* moving window with back-recover *is a set $W \subset B(0,c)$ that is shifted at each iteration in the direction of the movement of the machine $\hat{d} \in \mathcal{D}$, and reconstructed in direction u by the following function.*

$$\begin{array}{rccl} {}^{u}H_c^{\hat{d}}: & \mathcal{P}(B(0,c)) & \longrightarrow & \mathcal{P}(B(0,c)) \\ & W & \longrightarrow & {}^{u}H_c^{\hat{d}}(W) = [W - \hat{d} - \mathbb{N}u] \cap B(0,c) \end{array}$$

Lemma 6.2 *Let us suppose that the machine is following a regular trajectory $(x_i)_{i\in\mathbb{N}}$ of period $p < r$ and drift vector v such that $||v||_1 \leq r$. Let us consider a moving window with back-recover, initialised as $W_0 = (S - x_0 - \mathbb{N}u) \cap B(0,c)$, where $S \subseteq B(x_0, r)$, $c \geq 4r$ and $||u||_1 \leq r$, and it is updated through the function $W_{i+1} = {}^{u}H_c^{d_i}(W_i)$, where $d_i = x_i - x_{i-1}$ corresponds to the head movement at iteration i. Then we can assert, for each i, that either:*

1. *there exists $k \leq i$ such that $(S - \mathbb{N}u) \cap B(x_k, 2r) = \emptyset$, and x_j will never be in $S - \mathbb{N}u$ for no $j \geq k$, or*
2. *for each $k \leq i$, $(S - \mathbb{N}u) \cap B(x_k, 2r) \neq \emptyset$ and $(W_k \cap B(0, c-r)) + x_k$ contains exactly all the cells of $S - \mathbb{N}u$ at a distance less than $c - r$ from x_k.*

Proof: Let us first remark that the hypothesis of Lemma 5.3 holds by taking $w = x_j$ for each $j \in \{0, .., p-1\}$. Second, we remark that $x_i \in x_j + \mathbb{N}v \subseteq B(x_0, r) + \mathbb{N}v$ for some $j \in \{0, .., p-1\}$. We will prove the present lemma by induction on i. For $i = 0$, the lemma is true by definition: $(S - \mathbb{N}u) \cap B(x_0, 2r) \neq \emptyset$ and $(W_0 + x_0) \cap B(x_0, c) = (S - \mathbb{N}u) \cap B(x_0, c)$. Let $i \geq 1$.

1. If $(S - \mathbb{N}u) \cap B(x_k, 2r) = \emptyset$, for some $k \leq i$, we deduce that the distance between $S - \mathbb{N}u$ and x_k is greater than $2r$. Then by Lemma 5.3, we conclude that the machine will not visit the set $(S - \mathbb{N}u)$ any more.

2. If $(S - \mathbb{N}u) \cap B(x_k, 2r) \neq \emptyset$, for every $k \leq i$, we can assure that x_k is at a distance less than or equal to $2r$ from $S - \mathbb{R}_+ u$.

   On the other hand, $(W_k + x_k) \cap B(x_k, c - r) \subseteq (S - \mathbb{N}u) \cap B(x_k, c - r)$ for every k. By the induction hypothesis, the converse inclusion holds for $k < i$. Let us prove it for $k = i$.

   Since $p < r$, we can assume that $i \geq p$. Let $s \in S$ and $n \in \mathbb{N}$ be such that $s - nu \in B(x_i, c - r)$. We have to prove that $s - nu \in W_i + x_i$.

   From the hypothesis, there exists some $m \in \mathbb{N}$ such that $s - mu \in B(x_i, 2r)$. If $m \leq n$ we are done, because $s - mu \in B(x_{i-1}, c - r)$ and, by the induction hypothesis, it is in $W_{i-1} + x_{i-1}$, and then $s - nu$ is added to $W_i + x_i = (W_{i-1} + x_{i-1} - \mathbb{N}u) \cap B(x_i, c)$.

   If $m > n$, let $j \in \{0, .., p - 1\}$ and $t \in \mathbb{N}$ such that $x_i = x_{j+tp} = x_j + tv$. Since $s \in B(x_0, r)$ and $s - mu \in B(x_j + tv, 2r)$, by convexity, we can assure that there exists $l \in [0, t]$ such that $s - nu \in B(x_j + lv, 2r)$, and then there exists $l' \in \{0, .., t - 1\}$ such that $s - nu \in B(x_j + l'v, 3r)$. Therefore, by the induction hypothesis, $s - nu \in W_{j+l'p} + x_{j+l'p}$.

   From iterations $j + l'p$ to $j + tp = i$ the machine stays within a radius r from the segment $[x_j + l'v, x_i]$ which, by convexity again, is contained in $B(s - nu, c - r)$. This means that $s - nu$ stays within a radius c from the machine. Thus, it never exits the moving window, *i. e.*, $s - nu \in W_i + x_i$.

□

From these last lemmas, if we take U and V as moving windows or moving windows with back recover of radius $c = 4r$, they will contain the correct information in a radius $3r$.

6.3 Using the counters

The idea of storing *big* vectors is to write them in the form $d + Nv$, where $N \in \mathbb{Z}$ and d is a *small* vector. This representation is not unique, thus we will introduce several definitions and procedures to compute it in real time.

Definition 6.3 *Given two vectors u and v, we define the function $p(u, v)$ that gives the pair (d, n) where n is the integer that minimises the value $||u - nv||_1$ and $d = u - nv$. If any ambiguity exists, two criteria are followed:*

- *If v is parallel to $(1, 1)$ or $(1, -1)$, then n is chosen to minimise $||u - nv||_2$.*
- *If two integers fit the former criteria, then we choose the one of the smallest size.*

At the same time, we define the function $p'(u, v)$ which does the same but choosing $n \in \{-1, 0, 1\}$.

Definition 6.4 *In order to keep a sequence of positions $(x_i)_{i \in \mathbb{N}}$, where $x_0 = 0$, we start by initialising the counter as $n_0 = 0$ and the vector as $d_0 = 0$. Then, when the head of the machine moves in direction $z_i = x_i - x_{i-1}$, we proceed as follows:*

- *compute $(d_i, m_i) = p'(d_{i-1} + z_i, v)$,*

- *assign* $n_i = n_{i-1} + m_i$.

Proposition 6.1 *If the counter is initialised and updated through the procedure defined in definition 6.4, then at each iteration we have:*

1. $d_i = x_i - n_i v$, *and*
2. $(d_i, n_i) = p(x_i, v)$.

Proof: By induction on i. It is clear for the first iteration. Let us suppose it true for iteration i.

Assertion 1) is clear because, by definition, $d_i = d_{i-1} + z_i - m_i v = x_{i-1} - n_{i-1}v + x_i - x_{i-1} - m_i v = x_i - (n_{i-1} + m_i)v = x_i - n_i v$.

Assertion 2) is less clear because the updating is done through p' and not p. The key point to prove this is that $z_i \in B(0,1)$. We divide the analysis in two cases.

Case 1 v *is parallel either to* $(1,1)$ *or* $(1,-1)$. In this case, p just projects u on the line $\mathbb{R}v$ to obtain xv and then it takes the better integer approximation of x. Projecting $x_{i-1} + z_i$ is equivalent to projecting $d_{i-1} + z_i$, and since the projection is a linear function, this is equivalent to projecting z_i and adding it to d_{i-1}. In this case, the projection of z_i over v has norm smaller than 1. Thus $p(d_{i-1} + z_i, v)$ will give a pair (d, m), where m is an integer between -1 and 1, then $p(d_{i-1} + z_i, v) = p'(d_{i-1} + z_i, v)$.

Case 2 v *is neither parallel to* $(1,1)$ *nor* $(1,-1)$. In this case, we prove first that for every point u, there are at most two integers that minimise $||u - mv||_1$. In fact, it is not difficult to see that the set of points that are at the same distance, in norm 1, from $(0,0)$ and v is the union of two parallel semi lines, either vertical or horizontal, and one diagonal segment as shown in figure 4. The set of points equidistant from v and $2v$ is a parallel translation of the former set, thus they have no intersections, which proves that u can not be equidistant from more than two points in $\mathbb{Z}v$. The result is concluded by noticing that, since $v \in \mathbb{Z}^2$, the distance between these sets is at least 1. Then, by moving through a vector $z_i \in B(0,1)$ it is impossible to transverse two of these frontiers at the same time. Then if d_{i-1} is already the nearest vector to $\mathbb{Z}v$ at $n_{i-1}v$, then the nearest point of $\mathbb{Z}v$ to $d_{i-1} + z_i$ will be either $(n-1)v$, nv or $(n+1)v$, thus $p(d_{i-1} + z_i, v) = p'(d_{i-1} + z_i, v)$ as before.

□

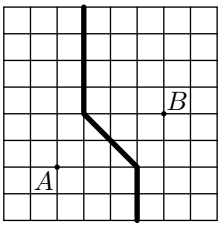

Fig. 4: The bold line represents the set of points that are at equal distance from A and B.

From the last proposition, it is clear that $x_i = 0$ if and only if $d = 0$ and the counter is 0. This gives us a way to detect when $x_i = 0$, which is useful to detect if the machine is over the pebble again. Also, if the counter is 0, we know that the machine is over a line that passes through the stored point. In the case this position is the point T where the machine exited the ball of radius $3r$ of the origin, passing through this line indicates us that the machine can be near the initial part of its trajectory and, thus, information contained in the window V_0 should be considered.

This procedure works well, except if the vector d becomes too large, the automaton has a finite memory, and we need to limit the size of d. Initially the vector v is chosen as the periodic net movement of the head. In this way, the size of d is naturally bounded by r. But if the machine finds a pebble, its movement will change, eventually adopting a new periodic drift u. If this is the case and the vector d becomes bigger than $2r$, then, by lemma 5.3, x_i will never be 0, because 0 is on the line $\mathbb{Z}v$ (except if another pebble is found; but in that case we enter stage 3 and we can forget the initial point).

The defined automata recognises the t-shift of the machine for which it was defined because it maintains true and sufficient information about the configuration that produced the input trajectory.

Theorem 6.1 *Given a 2-pebble machine M, there exists a 2-counter machine that recognises L_M in real time.*

7 Concluding remarks

From Delorme and Mazoyer we already knew that 0 and 1-pebble machines were equivalent. The work of Gajardo shows that their t-shift is sofic. From the present work we know that, when 2 pebbles are available, a 2-counter machine is enough to recognise its t-shift.

One can imagine that a machine with 3 pebbles will need 3-counters; unfOrtunately, the present construction cannot be adapted to that case. The main reason is that, with 3 pebbles, the machine can make triangular tours of arbitrary size, and we cannot use lemma 5.3 to discard a position when it goes too far from the line of movement of the machine. In a first approach, we can say that 6 counters are enough: two counters to record the coordinates of each pebble position. But the phase when less than 3 pebbles have been found, requires to record the whole set of visited cells which can be big and difficult to code.

If more than one symbol is allowed, even 0-pebble machines are outside the real time hierarchy, as it is shown in Gajardo and Mazoyer (2007). Nevertheless, particular machines can have lower complexity. In Gajardo and Guillon (2010), one-head machines in the 1-dimensional grid are studied, showing that machines whose t-shift is recognised with a 1-pushdown automaton have very restricted movement abilities: they cannot make "zigzags". Since zigzags are the most complex movements that a 2-pebble machine can do, we can imagine that 2-pebble machines without zigzags behave as 0-pebble machines. Is this true? Are there any 2-pebble machines whose t-shifts are not sofic but recognised with a 1-pushdown automaton?

Our construction is strongly based on some simple properties of lines and balls in $\mathbb{Z}^2$. But these properties are not due to the plane structure of this grid. Then, our results are still valid if the machine is defined on the d-dimensional grid $\mathbb{Z}^d$. Are they valid over the Cayley graphs of other groups?

References

S. Aanderaa. On k-tape versus (k-1)-tape real time computation. In R. Karp, editor, *Complexity of computation*, volume 7 of *SIAM-AMS Proceedings*, pages 75–96. Amer. Math. Soc., 1974.

M. Blum and C. Hewitt. Automata on a 2-dimensional tape. In *Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, FOCS '67, pages 155–160, Washington, DC, USA, 1967. IEEE Computer Society.

L. A. Bunimovich and S. Troubetzkoy. Recurrence properties of Lorentz lattice gas cellular automata. *J. of Stat. Physics*, 67:289–302, 1992.

L. A. Bunimovich and S. Troubetzkoy. Rotators, periodicity, and absence of diffusion in cyclic cellular automata. *J. of Stat. Physics*, 74:1–10, 1994.

M. Delorme and J. Mazoyer. Pebble automata. figures families recognition and universality. Technical Report 32, Ecole Normale Supérieure de Lyon, Lyon,France, 1999.

P. Dorbec and A. Gajardo. Langton's fly. Technical Report 07, Departamento de Ingeniería Matemática, Concepción, Chile, 2007.

P. Fischer, A. Meyer, and A. Rosemberg. Counter machines and counter languages. *Mathematical systems theory*, 2:265–283, 1968.

A. Gajardo. Sofic one head machines. In B. Durand, editor, *Journées Automates Cellulaires*, pages 54–64, 2008.

A. Gajardo and P. Guillon. Zigzags in turing machines. In *Lecture Notes in Computer Science*, volume 6072, pages 109–119, 2010.

A. Gajardo and J. Mazoyer. One head machines from a symbolic approach. *Theor. Comput. Sci.*, 370: 34–47, 2007.

A. Gajardo, J. Kari, and A. Moreira. On time-symmetry in cellular automata. *Journal of Computer and System Sciences*, 78:1115–1126, 2012.

J. M. F. Gunn and M. Ortuño. Percolation and motion in a simple random environment. *J. Phys. A.: Math. Gen.*, 18:L1095–L1101, 1985.

J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2001.

P. Kůrka. On topological dynamics of Turing machines. *Theor. Comput. Sci.*, 174:203–216, March 1997. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(96)00025-4. URL `http://dx.doi.org/10.1016/S0304-3975(96)00025-4`.

P. Kůrka. *Topological and Symbolic Dynamics*. Société Mathématique de France, Paris, France, 2003.

P. Kůrka and A. Maass. Realtime subshifts. *Theor. Comput. Sci.*, 237:307–325, 2000.

J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

K. Wagner and G. Wechsung. *Computational complexity*. VEG Deutscher Verlag der Wissenschaften, Berlin, 1986.