Two-way non-deterministic finite automata with a write-once track recognize regular languages only

Berke Durak
Université Paris VII, L.I.A.F.A.
2 place Jussieu, 75221 Paris, France
durak@liafa.jussieu.fr

Abstract

The basic finite automata model has been extended over the years with different acceptance modes (non-determinism, alternation), new or improved devices (two-way heads, pebbles, nested pebbles) or with cooperation. None of these additions permits recognition of non-regular languages. The purpose of this work is to investigate a new extension which is inspired by an extension of 2DPDAs. Mogensen enhanced these with what he called a WORM track and showed that Cook's linear-time simulation algorithm still holds. Here we trade the pushdown store for non-determinism and show that the languages of these new types of finite automata are still regular. These machines are also interesting because they allow us to describe certain languages with fewer states, and also because all our attempts at further extending this model cross the boundary of regular languages. While similar to Hennie machines, our model does not require a time bound on its computations.

1 Introduction

Two-way deterministic pushdown automata (2DPDAs) have played an important role in the development of formal language theory [7]. It is a well-known fact [3, 15] that the class of languages recognizable by multihead (or single-head with polynomial padding) 2DPDAs is strongly equal to P in the sense that the polynomial exponent is related to the number of heads. By Cook's result [4], a k-head 2DPDA can be simulated on a random-access machine in time $O(m^k)$ where m is the length of the input. This has inspired some interesting algorithms such as the Knuth-Morris-Pratt [12] algorithm or a linear-time algorithm for recognizing "PALSTAR" [8].

In [14] the idea of extending 2DPDAs with a special kind of track, called a write-once read-many (WORM) track was introduced. This was done with the hope of increasing their power while retaining their linear-time simulation property. Basically, these tracks are working like classical Turing machine tracks, except that reading a blank square causes it to be filled with a special symbol *prior to the access*, that symbol being returned as the result of the read instruction; also, writing to a non-blank square causes nothing. Surprisingly, Cook's algorithm, slightly modified, can simulate these WORM machines in linear time. This allows us, for instance, to perform lexical tokenization in linear time [17],

as opposed to the quadratic worst-case time of current lexical scanners. WORM tracks are also useful for expressing some languages more easily, such as $\{uu^Rvv^R | u, v \in \{a, b\}^*\}$ (PALSQUARE). However it is still an open question whether or not WORM-2DPDAs recognize more languages than 2DPDAs. It seems natural to investigate the simpler case of a standard finite automaton provided with a WORM track. This gives a two-way deterministic automaton with a WORM track (a WORM-2DFA) which is easily shown to accept regular languages only. But if one introduces non-determinism (thus obtaining what we call WORM-2NFAs), the regularity of the recognized languages is no longer trivial, and is the main result of this article.

Our contribution is in the vein of improving how far we can go by enriching the natural finite state model while keeping the same expressive power. Here we think of models such as one-way or two-way, deterministic, non-deterministic or alternating automata, which can be cooperating or which can have one pebble or a number of nested pebbles [9]. All these recognize regular languages.

WORM-2NFAs are somewhat similar to non-deterministic Hennie machines. These are single-head Turing machines whose heads do not leave the input portion of their tape, and which have the bounded visit property, that is, there is a constant c such that the machine never visits any given position more than c times [1]. These machines recognize regular languages only. In Hennie's original paper [11] it was shown that deterministic linear-time Turing machines have the bounded visit property. It should be noted that there exists linear-time non-deterministic Turing machines recognizing non-regular, NP-complete languages[1] [13]. Since they are allowed to modify their input tapes, Hennie machines can also be seen as transducers. However their direct transductions are not necessarily rational[2]. Furthermore, the linearity of running time is a non-trivial and thus undecidable property of Turing machines, making the class of Hennie machines non constructive. Hence WORM-2NFAs constitute an interesting class of non-deterministic tape machines which can run in unbounded time and whose languages are regular. They may in fact be more succint machines since they can solve SAT using a polynomial number of states.

2 The model

A k-WORM-2NFA is a 2NFA having, on its tape and below its input track, k WORM tracks. All these tracks have the same length, and all are accessed with the same input/output head. The cells of the WORM tracks are called WORM squares or WORM cells. Initially, a WORM square is blank (we denote blank squares by $\square$). If the automaton decides to write a symbol $\gamma \in \Gamma$ to a blank square, its content becomes γ and can no longer change. Further writes will silently be ignored – in particular, they will not cause the computation to abort. Reading a non-blank square returns the symbol it contains. However, reading a blank square causes that square to be filled with a special symbol $\theta \in \Gamma$, which is returned as the result of this and of all further read attempts. Therefore, the automaton can never observe blank squares and has no way of discerning a square that has been containing θ for a long time from a square that was blank until the very

[1] These do not verify the bounded visit property and thus are not Hennie machines.

[2] Their inverse transductions are rational, see [5].

moment it has been read. A WORM track can thus be seen as a cache for memorizing intermediate computing results as in dynamic programming: its behaviour ensures that any two reads from the same square will return the same result.

Formally a k-WORM-2NFA is an octuple $M = (Q, \Sigma, \Gamma, q^0, q^+, \tau, T, \kappa)$. The input alphabet is Σ. The WORM track alphabet Γ contains a special symbol θ. The set Q of states is divided into three disjoint subsets by the map $\tau : Q \to \{C, R, W\}$. States q such that $\tau(q) = C$ are control states (C-states) and cannot access the WORM tracks. States where $\tau(q) = W$ write on the WORM tracks (W-states). States where $\tau(q) = R$ read from the WORM tracks, but we recall that reading from a blank square causes that square to be filled with θ, so that R-states may actually modify the contents of the WORM tracks. The function κ maps W-states and R-states to the set $\{1, 2, \ldots, k\}$ and selects the WORM track to use by number; it can be omitted when $k = 1$. The initial state q^0 and the final state q^+ are C-states.

Transitions can be of two forms, depending on the type of the state they apply to. For $q, q' \in Q$ with $\tau(q) = C$, they are of the form (q, σ, d, q') where $\sigma \in \Sigma$ is a letter to be read from the input track and $d \in \{-1, 0, 1\}$ gives the head movement (left, no movement or right). For $q, q' \in Q$ with $\tau(q) = R$ or $\tau(q) = W$, they are of of the form (q, γ, q'), where $\gamma \in \Gamma$ gives a letter to read or write from one of the WORM tracks. The set of transitions of M is T. In all cases, we define $\alpha(t) = q$ to be the source and $\omega(t) = q'$ to be the target states of the transition t. We also require T to contain all transitions of the form (q^+, σ, d, q^+) (for $\sigma \in \Sigma$ and $d \in \{-1, 0, 1\}$) and no transitions of the form (q^+, σ, d, q) with $q \neq q^+$.

Definition 1 (Configurations). *A* configuration *of M over an input u of length m is a triple (q, i, w) where $q \in Q$ is the state, $1 \leq i \leq m$ is the position and $w = (w_1, \ldots, w_k)$ are the WORM contents. The initial configuration is $(q^0, 1, (\square^m)^k)$. A configuration is* final *whenever $q = q^+$.*

Each transition defines a partial function on the set of configurations. That is, when applicable, a transition leads from one configuration to exactly another configuration.

Definition 2 (Effects of transitions). *Let t be a transition and $c = (q, i, w)$ be a configuration. The* effect *of t on c is a new configuration written $c \cdot t$ and is defined as follows :*

- *When $t = (q, \sigma, d, q')$ with $\tau(q) = C$, and if $1 \leq i + d \leq m$ holds, we have $c \cdot t = (q', i + d, w)$; otherwise, $c \cdot t$ is undefined.*
- *When $t = (q, \gamma, q')$ with $\tau(q) = R$, the i-th square of the $\kappa(q)$-th WORM track is read. Depending on the blankness of that square and the value of θ, we have three cases:*
  - *When the square is not blank, if it contains the symbol γ, the transition is applicable and $c \cdot t = (q', i, w)$. If it contains a symbol other than γ, the transition is not applicable.*
  - *When the square is blank and $\gamma = \theta$, θ is first written into the square. Let w' stand for the contents of the k WORM tracks where the aforementioned square has been filled with θ. The transition is applicable and $c \cdot t = (q', i, w')$.*

– *Otherwise, that is when the square is blank and $\gamma \neq \theta$, the transition is not applicable. Intuitively, the square gets filled with θ, but M is not expecting to read a θ and blocks.*

- *When $t = (q, \gamma, q')$ with $\tau(q) = W$, γ is written to $w_{\kappa(q)}[i]$ if it is blank, giving w'; in that case $c \cdot t = (q', i, w')$. Otherwise only the state is changed and we have $c \cdot t = (q', i, w)$.*

Definition 3 (Runs, computations, language). *A* run *is a finite sequence $t_1 t_2 \cdots t_n$ of transitions. A* computation *(over the input u) is a run $t_1 \cdots t_n$ that can be applied to the initial configuration $c^0 = (q^0, 1, (\square^m)^k)$, that is: t_1 has a defined effect on c^0, t_2 has a defined effect on $c_0 \cdot t_1$ and so on such that $c^0 \cdot t_1 \cdot t_2 \cdots t_n$ has a defined value $c = (q, i, w)$. The run then leads from c^0 to c. Such a computation is* accepting *when the obtained configuration is final, that is when $q = q^+$. The language $\mathscr{L}(M)$ of M is the set of input words u admitting an accepting computation.*

The functions α and ω readily extend to runs: $\alpha(t_1 \cdots t_n) = \alpha(t_1)$ and $\omega(t_1 \cdots t_n) = \omega(t_n)$ when $n \geq 1$. We now give an example showing the usefulness of WORM tracks.

Example 1. *In [2] the set* Collage(L) *of collages of a language L was defined as the set of words that can be obtained by pasting one on top of the other at random positions arbitrary words chosen from L. (In fact this idea was inspired by WORM tracks.) See for instance figure 1(a). If L is a regular language recognized by a one-way non-deterministic automaton M having n states then it is very easy to give a $O(n)$-state 1-WORM-2NFA recognizing* Collage(L)*: the automaton repeatedly places its head at a random position and writes an arbitrary word from L over its WORM track by randomly following transitions of M (fig. 1(b), 1(c)). (The automaton guesses the words from top to bottom.) It then checks that its WORM and input tracks have the same contents, see fig. 1(d).*

Besides being a computational device, WORM tracks can naturally be used as output devices. Hence a WORM-2NFA can be seen as a transducer. This is valuable, since Hennie machines apart, all known regular transducers with a two-way input head have only a one-way, pushdown-store-like output [6].

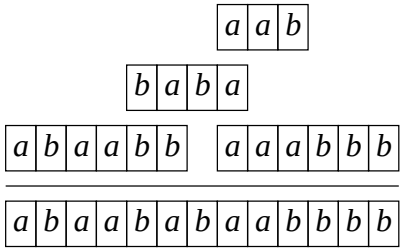
3 The membership problem

The membership problem for a class $\mathscr{C}$ of automata is defined as the language

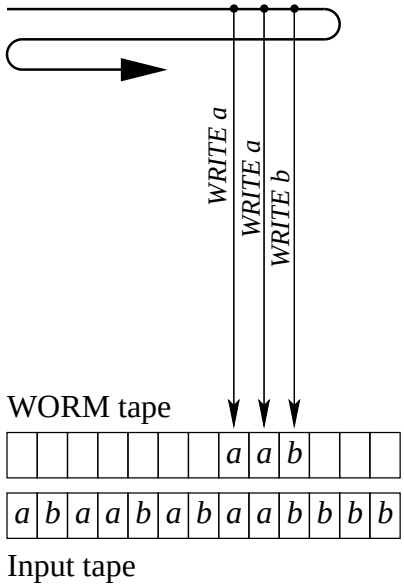
$$L = \{\langle M \rangle \# u \mid M \in \mathscr{C} \text{ and } u \text{ is accepted by } M\}$$

where $\langle M \rangle$ is a suitable encoding of M. The complexity of L thus depends on the encoding used. Non-deterministic automata are conveniently encoded as the list of their transitions; this way their descriptional size is at most quadratic in their number of states. We adopt the same convention and define the size of a WORM-2NFA as the number of its transitions.

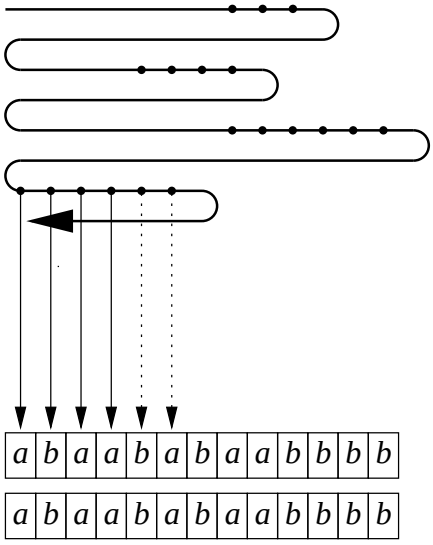
Since a WORM-2NFA can have computations of arbitrary length, it is not apparent that the membership problem is even decidable. However, the following lemma gives an upper bound on the length of computations.



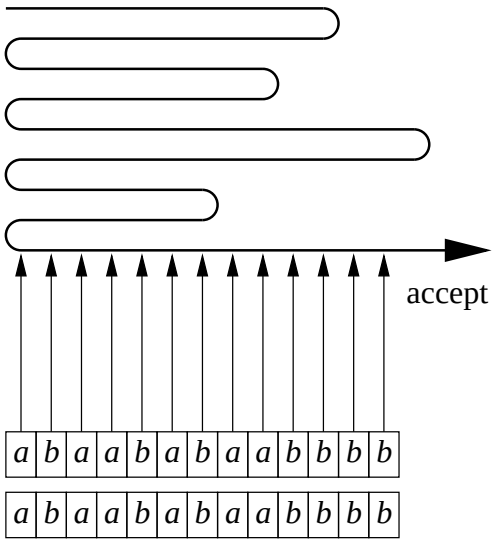
(a) A collage of the words *aab*, *baba*, *abaabb* and *aaabbb* giving *abaababaabbbb*.



(b) The WORM-2NFA M guesses the topmost word *aab* and writes it.



(c) The machine continues by guessing the other words of the collage and writing them to the WORM track.



(d) Finally M compares the contents of its WORM and input tracks.

Figure 1: How a WORM-2NFA can recognize Collage(L).

Lemma 1. *If a word u is accepted by a k-WORM-2NFA M having m states, then it has an accepting computation of length $O(km|u|^2)$.*

Sketch. By an easy application of the pumping lemma every computation of a 2NFA is equivalent to a computation having its size linear in the product of the number of states and the length of the input. As M behaves as a 2NFA between two modifications of the WORM track, and since there cannot be more modifications than WORM squares, the upper bound follows. □

Actually, the membership problem is NP-complete. Using an idea similar to [16], for every $n \geq 1$ we encode Boolean circuits having n gates over the alphabet $\Sigma_n = \{x_1, x_2, \ldots, x_n, \wedge, \vee, \neg, I, \#\}$ as follows. Such a circuit is made of input and internal gates. The i-th gate of the circuit is encoded as a factor of the form $\#x_i I$ (for input gates), $\#x_i \neg x_j$ (for a NOT gate negating the output of gate j), $\#x_i \wedge x_j x_k$ or $\#x_i \vee x_j x_k$ (for an AND or an OR gate having gates j and k as inputs). Gate labels $x_1, \ldots, x_n$ following the # symbol are called *defining occurrences*. The circuit is satisfiable when there is a way to assign truth values to the input gates such that the value computed for x_1 is true. Let S_n^+ be the set of encodings of satisfiable circuits having $\leq n$ gates. A WORM-2NFA can then guess values for the input and the internal gates and check that it satisfies the circuit:

Lemma 2. *For every n there is an $O(n^3)$-state WORM-2NFA M_n accepting S_n^+, and a description of M_n can be computed in time polynomial in n.*

In fact, the machines M_n are no more than two-way deterministic automata with a length-preserving homomorphism [1]: once the values have been guessed, the WORM track is not written to any more.

Proposition 1. *The membership problem for WORM-2NFAs is NP-complete.*

Proof. Let (M, u) be an instance of the membership problem, where M is a k-WORM-2NFA having m states. By lemma 1, u is accepted by M if and only if there is a valid list of $\geq 4km|u|^2$ transitions. Such a list can be guessed by a non-deterministic Turing machine and checked for acceptance in time polynomial in the size of M, showing that the problem is in NP. Conversely, an n-gate instance of circuit satisfiability can be easily converted by lemma 2 into an $O(n^3)$-sized instance of the membership problem. This shows NP-hardness. □

4 Regularity

Given a computation c over an input uv, consider the instants where the head of the automaton crosses the boundary between the left half u and the right half v. (See fig. 2(a)) These are the times t where the head is at position $p = |u|$ and at position $p + 1$ at the next moment $t + 1$, or at position $p + 1$ and at position p at the next moment. In the former case the automaton exits the left half and enters the right half in some state q_1. This can be seen as the left half querying the right half, the question being the state q_1. The answer to that query is given when the automaton exits the right half and comes back to the left half in some state q_2. (The automaton can also accept in the right half, which we count as the answer q^+.) This answer can itself be seen as a query from the

right half to the left half, and so on. This sequence of queries/answers is a dialog between the two halves, and each half is a speaker. This concept is well-known under the name of "crossing sequences" [11]; it comes naturally when one tries to show the regularity of 2NFA or 2DFA languages.

It should be noted that, as for 2NFA and 2DFAs, only the state information crosses the boundary between u and v. This is because the set of WORM squares of the two halves are disjoint. The two important ideas in the proof of the regularity of $\mathscr{L}(M)$ are then as follows. First, a speaker is defined by its observable properties, that is, the way it answers. This allows us to sidestep the problem of the unbounded information content of WORM tracks. (A similar approach is used for pebble-2NFAs.) Second, the information useful for proving the regularity can be finitely encoded. With these we are able to show that $\mathscr{L}(M)$ has a finite number of left residuals.

Definition 4 (Splicings). *Let $c = t_1 \cdots t_n$ be a computation. The* splicing *of c at position p ($1 \leq p \leq m$) is a factorization $c = a_1 a_2 \cdots a_r$ of c into runs $a_1, \ldots, a_r$ such that odd-numbered runs proceed in the left half (at positions $\leq p$) and even-numbered runs proceed in the right half (at positions $> p$).*

Our requirement of inclusion for transitions of the form (q^+, γ, d, q^+) ensures that we can restrict ourselves to accepting computations which finish by having the head scan the input from left to right in state q^+: any splicing of such a computation will finish in the right half and thus have an even number of blocks.

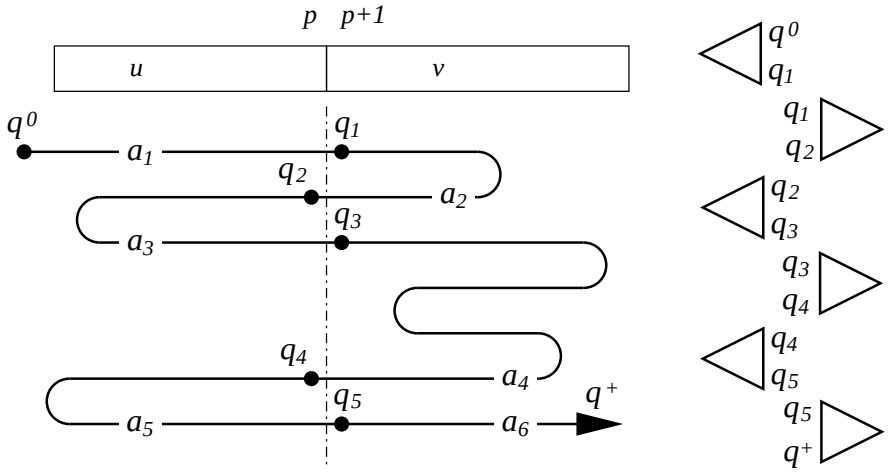
Definition 5 (Triangles, dialogs). *Given a computation c spliced at p as $c = a_1 a_2 \ldots a_r$, we may examine the exchange of states between the left and right halves, that is, between the odd- and even-numbered runs. For odd j the automaton leaves the left run a_j and enters the right run a_{j+1} in state $q_1 = \omega(a_j) = \alpha(a_{j+1})$. The right run a_{j+1} then returns to the left run in state $q_2 = \omega(a_{j+1})$. Thus the run a_{j+1} leads from q_1 to q_2 in the right half. The information carried by the run a_{j+1} is reduced to the pair of states q_1, q_2 along with an indication of which part the run is executed in. This gives a symbol $\tilde{a}_{j+1} = [q_1 \triangleright q_2]$. (If a_{j+1} was instead a left run we would write $\tilde{a}_{j+1} = [q_1 \triangleleft q_2]$.) The* dialog *of the computation c at p is then the sequence of triangles of the runs of c spliced at p:* $\mathrm{dlg}_p(c) = \tilde{a}_1 \tilde{a}_2 \cdots \tilde{a}_r$.

In figure 2(a) we have a computation c over a word uv spliced at position $|u| = p$ as $c = a_1 a_2 \cdots a_6$. The left run a_1 leads from the initial state q^0 to q_1. The computation then continues in the right half with the run a_2 leading from q_1, back to the left half in state q_2, and so on. The dialog of c at $|u|$ is therefore the sequence of triangles (see figure 2(b)):

$$\mathrm{dlg}_{|u|}(c) = [q^0 \triangleleft q_1][q_1 \triangleright q_2][q_2 \triangleleft q_3][q_3 \triangleright q_4][q_4 \triangleleft q_5][q_5 \triangleright q^+]$$

Take two accepting computations c and c' over uv and wx respectively. It has already been noted [11] for Turing machines that if c and c' have the same dialog at positions $|u|$ and $|w|$, respectively, then we can "join" c and c' into an accepting computation c'' for ux. Indeed, assume $\mathrm{dlg}_{|u|}(c) = \mathrm{dlg}_{|w|}(c')$ and let $c = a_1 \cdots a_k$ and $c' = b_1 \cdots b_k$ be splicings of c and c' at $|u|$ and $|w|$ respectively. Then $c'' = a_1 b_2 a_3 b_4 \cdots b_k$ is a valid computation for ux.

We now examine a very particular property of WORM-2NFAs which is the key of our proof. Let c be a computation over u. Initially, all k WORM tracks are empty. As



(a) A computation c over the word uv, spliced at $|p|$.

(b) The dialog of c.

Figure 2: The dialog of a computation.

the computation goes on, WORM squares get filled either by fortunate write operations or unfortunate read attempts. At the end of the computation the j-th WORM track contains the word $w_j \in (\Gamma \cup \{\square\})^m$ where $m = |u|$. Assume we restart the machine M from the configuration $(q^0, 1, w)$. In other words, we restart M in the initial state, with its head at the initial leftmost position, but the WORM tracks already filled instead of being blank. The set of possible computations starting from that configuration will be different from the set of computations starting with blank tracks: some computations will require different WORM contents and will no longer be possible, whereas other computations reading previously blank squares will be enabled. However the computation c will still be possible. Indeed, for every WORM square, look at the first access to that square in c. If it is a write access, M will try, in the second run of c over the already filled WORM tracks, to write γ again to that square. This will pose no problem as rewriting is allowed. If it is a read access, the square will contain θ; since the computation c was successful, M was satisfied reading a θ from that square, and will also be satisfied in the second run. Further accesses happen in the same way. Generalizing this observation, we obtain the following lemma:

Lemma 3 (Repeatability). *Let M be a k-WORM-2NFA over an input u. If there is a run $a = t_1 \cdots t_n$ leading from (q, i, w) to (q', i', w'), and if w'' are WORM contents that can be obtained by filling blank squares of w', then the run a also leads from (q, i, w'') to (q', i', w'').*

Consider again two computations c and c' over uv and wx spliced at $|u|$ and $|w|$ as $c = a_1 \cdots a_k$ and $c' = b_1 \cdots b_\ell$. We will see that the repeatability lemma allows us to join c and c' on a condition over $\mathrm{dlg}_{|u|}(c)$ and $\mathrm{dlg}_{|w|}(c')$ much weaker than equality: c and c' will be joinable if the two dialogs have the same subsequences of new triangles. These are

defined as follows.

Definition 6. *Let X be a set; X^* stands for the set of finite sequences over X. We recursively define the operator $I : X^* \to X^*$ as follows:*

$$I(\varepsilon) = \varepsilon$$
$$\forall x \in X, \forall y \in X^* \quad I(yx) = \begin{cases} I(y)x & \text{if } x \text{ does not occur in } y, \\ I(y) & \text{otherwise.} \end{cases}$$

(Here ε denotes the empty word.) The sequence $I(x)$ is called the novelty sequence *of x.*

This operator keeps only the first occurrence of each letter, deleting repeated letters. Thus if $X = \{0, 1, 2, \ldots, 9\}$ we have $I(314159265358) = 31459268$.

We can now give the core lemma of our result.

Lemma 4. *Let $X = \{[q \triangleleft q'], [q \triangleright q'] \mid q, q' \in Q\}$ be the set of possible triangles. If c and c' are two accepting computations over uv and wx respectively such that $D = D'$ where $D = I(\mathrm{dlg}_{|u|}(c))$ and $D' = I(\mathrm{dlg}_{|w|}(c'))$, then there exists an accepting computation c'' over ux.*

Proof. Let $c = a_1 \cdots a_k$ and $c' = b_1 \cdots b_\ell$ be splicings of c and c' at $|u|$ and $|w|$ respectively. Each run can read and modify the corresponding half of the WORM tracks. Therefore in each computation, runs working on the same half can depend one on another: a run x may require the ordered execution of some runs, called *prerequisites* of x, before being executable.

Due to the unbounded number of squares in each half, these dependencies can be arbitrarily complex. However their analysis is not necessary thanks to the repeatability lemma. Indeed once a run leading from q to q' has been executed, it can be executed again to get from q to q' with no other consequences, i.e., without further modifying the WORM tracks. This property allows us to classify the runs of a computation in two ways.

Definition 7 (Novel runs). *A run x is said to be* novel *(or a* novelty*) in a computation when no preceding run has the same triangle as x, that is, no previous run of the same half leads from the same starting state to the same ending state.*

Let $N \subseteq \{1, \ldots, k\}$ (resp. $N' \subseteq \{1, \ldots, \ell\}$) be the set of indices of novel runs in c (resp. in c'). Since $I(\mathrm{dlg}_{|u|}(c)) = I(\mathrm{dlg}_{|w|}(c'))$ it follows that the sets N and N' have the same cardinality n; let $\lambda_1 < \cdots < \lambda_n$ and $\lambda'_1 < \cdots < \lambda'_n$ be their respective elements. For every ν the runs a_{λ_ν} and $b_{\lambda'_\nu}$ are equivalent in the sense that they have the same triangle. We will join c and c' in n iterations. At the ν-th iteration, we will join the runs $a_{\lambda_\nu}, \ldots, a_{\lambda_{\nu+1}-1}$ and $b_{\lambda'_\nu}, \ldots, b_{\lambda'_{\nu+1}-1}$.

Definition 8. *The computations c and c' are* joinable up to the ν-th novelty *when there exists a computation c'' over ux such that $\alpha(c'') = q^0$, $\omega(c'') = \omega(a_{\lambda_\nu})$ and the following condition holds. Let $d_1 \cdots d_m$ be the splicing of c'' at position $|u|$. Set X to be set of runs of M, which we will now consider as individual symbols. By $\prod_{\substack{1 \le j \le m \\ j \text{ odd}}} d_j$ denote the word*

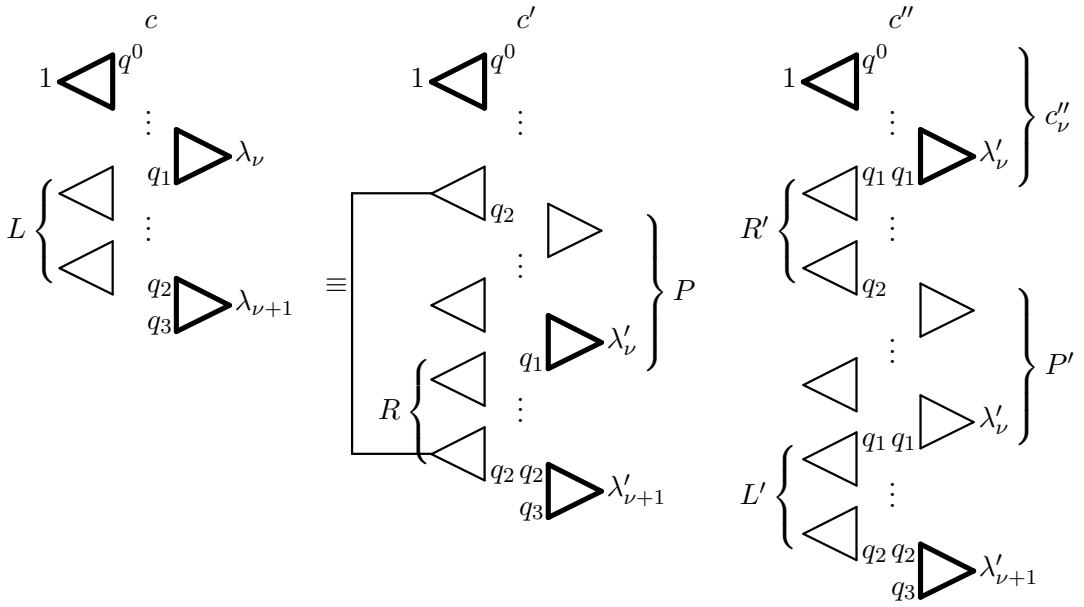


Figure 3: Joining two computations.

over X made of the left runs of d. We can then use the operator I over X^* as in definition 6 to give the condition:

$$I\left(\prod_{\substack{1\le j\le \lambda_\nu \\ j \text{ odd}}} a_j\right) = I\left(\prod_{\substack{1\le j\le m \\ j \text{ odd}}} d_j\right) \quad \textit{and} \quad I\left(\prod_{\substack{1\le j\le m \\ j \text{ even}}} d_j\right) = I\left(\prod_{\substack{1\le j\le \lambda'_\nu \\ j \text{ even}}} b_j\right) \tag{1}$$

This means that c'' is made of the left runs of c of indexes $\le \lambda_\nu$ and of the right runs of c' of indexes $\le \lambda'_\nu$, the ordering of the runs in c and c' being preserved.

We now show by induction that c and c' are joinable up to n. The first two runs of any computation are necesarily novel. Since $D = D'$, $c'' = a_1$ or $c'' = a_1 b_2$ are adequate computations over ux. Suppose that c and c' are joinable up to the ν-th novelty ($\nu \ge 2$). Let c''_0 be the resulting computation. It leads from $\alpha(a_1) = q^0$ to $\omega(a_{\lambda_\nu}) = \omega(b_{\lambda'_\nu}) = q_1$. Consider the next novel runs. In c this is $a_{\lambda_{\nu+1}}$ and it is equivalent in c' to (i.e., has the same triangle as) $b_{\lambda'_{\nu+1}}$. They both lead from $q_2 = \alpha(b_{\lambda'_{\nu+1}})$ to $q_3 = \omega(b_{\lambda'_{\nu+1}})$.

We now construct c'' by starting from c''_0. By symmetry, we only consider the case where the next novelty $\nu + 1$ is a right run. Between the last novelty ν and the next novelty $\nu + 1$, we have non-novel runs in c (call their sequence L) and in c' (call their sequence R): we can write $a = a_1 \cdots a_{\lambda_\nu} L a_{\lambda_{\nu+1}} \cdots a_k$ and $b = b_1 \cdots b_{\lambda'_\nu} R b_{\lambda'_{\nu+1}} \cdots b_\ell$.

For c'' to satisfy equation 1 for ν, the left runs of L and the right runs of R must be executed in order and then $b_{\lambda'_{\nu+1}}$ must be added. (See figure 3). Assume R is not empty (otherwise just skip this step). We need to modify R, which is a computation

over wx, into a computation over ux. To do this, first recall that all the runs of R are non-novel. By the inductive hypothesis, each left run of R has the same triangle as some left run of c_0''. By the repeatability lemma, we can replace each left run (over w) of R by an already executed equivalent left run (over u) of c_0''. This gives a computation R' over ux which does not further change the WORM contents of the left half, but behaves on the right half exactly as R over wx. This leads us into the state q_2 in the right half (since the $(\nu+1)$-th run is a right one). To be able to continue with a the symmetric substitution procedure on L, we need to get back into q_1 in the same half as ν (which can be left or right). Observe that since the left run $b_{\lambda'_{\nu+1}-1}$ is not novel in c_0'', there exists $s < \lambda'_\nu$ such that $\tilde{b}_s = \tilde{b}_{\lambda'_{\nu+1}-1}$. Therefore there exists a factor P of c_0'' leading from q_2 in the right half to q_1 in the corresponding half. As this factor has already been executed, we can reexecute it to get back into q_1. We are now in q_1 in the suitable half and we apply the symmetric substitution procedure on L by replacing the right runs of L with already executed, equivalent right runs of c_0''. This gives L'. Finally, we can add the next novelty $b_{\lambda'_{\nu+1}}$. The computations c and c' are thus joined up to the $(\nu+1)$-th novelty as $c'' = c_0'' \cdot R' \cdot P \cdot L' \cdot b_{\lambda'_{\nu+1}}$. (As noted above, there is a symmetric case where $\lambda_{\nu+1}$ is even.) This concludes the induction step, and thus the lemma. □

As an example consider the computations whose crossing sequences are:

$$c : 01\ 23\ 24\ 23\ 24\ 21\ 23\ 23\ 5 \quad c' : 01\ 23\ 24\ 23\ 24\ 21\ 24\ 23\ 23\ 5$$

Our construction gives a computation whose crossing sequence is:

$$c'' : 01\ 23\ 24\ 23\ 24\ 21\ 24\ 23\ 24\ 23\ 23\ 23\ 5$$

We are now ready to prove our main result.

Theorem 1. *Languages accepted by k-WORM-2NFAs are regular.*

Proof. Let M be be a k-WORM-2NFA as defined in section 2. To each word $u \in \Sigma^*$ we associate the set

$$F_u = \Big\{ I\big(\mathrm{dlg}_{|u|}(c)\big) \mid \exists c \exists v \;\; c \text{ is an accepting computation over } uv \Big\}.$$

We show that if $F_u = F_w$ for two words u and w then for all x, we have $ux \in \mathscr{L}(M)$ if and only if $wx \in \mathscr{L}(M)$.

By symmetry, it is enough to show one implication of the equivalence. Suppose there exists an accepting computation c' for wx. The sequence $D' = \mathrm{dlg}_{|w|}(c')$ is in F_u as well as in F_w. Therefore there is an accepting computation c for uv such that $\mathrm{dlg}_{|u|}(c) = D$, where v is some word. By lemma 4 there is an accepting computation for the word ux.

As the set of novelty sequences over triangles over Q is finite, there is only a finite number of possible values for F_u ($u \in \Sigma^*$). The left residual $u^{-1}.\mathscr{L}(M)$ being characterized by F_u, it follows that $\mathscr{L}(M)$ has at most as many left residuals as there are possible values for F_u and is thus regular. □

5 Concluding remarks

The regularity of WORM-2NFAs is very sensitive to the behaviour of the WORM cells. For instance, take a variant of WORM cells where any attempt to write to a non-blank cell causes the computation to halt. It is possible to accept a non-regular language such as $Z = \{a^m b^n \mid 1 \leq m \leq n\}$ with such cells: First check that the input is in a^+b^+. Then randomly select an a in the left half and mark it by writing X on the corresponding WORM cell. If the cell was already marked, the computation will abort. Now randomly select a b in the right half and check it. Continue as long as necessary. Finally, verify that all a's have been checked by reading them (reading the blank WORM cell under any a that was not checked will return θ which is obviously different from X), and then accept.

It is possible to define alternating WORM machines in the same way as alternating Turing machines are defined. Each subcomputation gets thus a copy of the WORM track which can be modified independently of the others. However this makes simulation of the "multiple writes disallowed" semantics possible. Indeed, as each branch of a universal fork has its own copy of the WORM track, a subcomputation can check if a given cell is blank without destroying the same cell in the parent's track and pass back that information.

Therefore, extending WORM-2NFAs while keeping regularity seems possible only under non-deterministic acceptance modes. However, while it is possible to add a pebble, or even multiple specially nested pebbles to 2NFAs or even 2AFAs and still remain in the realm of regular languages [10, 9], adding a single pebble to WORM-2NFAs permits recognition of the non-regular language Z.

A WORM-2NFA can easily recognize the image by a length preserving homomorphism of a 2DFA or 2NFA language by guessing a preimage on its WORM track and then executing the 2DFA or 2NFA. It has been shown [1] that a 2DFA with $O(n^2)$ states (resp. with $O(n)$ states) can simulate, under such an homomorphism, a 2AFA with n states (resp. a halting 2AFA with n states). It follows that an $O(n)$-state WORM-2NFA can simulate an n-state halting 2AFA or a $\sqrt{n}$-state classical 2AFA. Since 2AFAs are known [9] to be exponentially more succint than 2NFAs, it follows that WORM-2NFAs are at least exponentially more succint than 2NFAs.

The ability to hold a number of guessed values linear in the size of the input throughout the computation appears as an ability out of the reach of 2AFA, pebble-2AFA and other similar models. Also, compared to non-deterministic Hennie machines, WORM-2NFAs do not have a finite bound on the number of times they can visit a given square. We therefore conjecture that 2AFAs as well as non-deterministic Hennie machines cannot solve SAT with a polynomial number of states. We hope that these results will shed some light on the relative power of WORM-2DPDAs vs 2DPDAs.

Acknowledgements. I would like to thank Prof. Ch. Choffrut for his helpful advice and the many lengthy discussions about this subject.

References

[1] Jean-Camille Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

[2] Christian Choffrut and Berke Durak. Collage of two-dimensional words. *Theoret. Comp. Sci*, in press.

[3] Stephen A. Cook. Characterization of pushdown machines in terms of time-bounded computers. *Journal of the Association for Computing Machinery*, 18:4–18, 1971.

[4] Stephen A. Cook. Linear-time simulation of deterministic two-way pushdown automata. *Information Processing*, 71:75–80, 1972.

[5] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

[6] Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In Krzysztof Diks and Wojciech Rytter, editors, *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2002.

[7] Zvi Galil. Some open problems in the theory of computation as questions about two-way deterministic pushdown automata languages. *Mathematical Systems Theory*, 10:211–228, 1977.

[8] Zvi Galil and Joel Seiferas. A linear-time on-line recognition algorithm for "Palstar". *Journal of the Association for Computing Machinery*, 25:102–111, 1978.

[9] Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theor. Comput. Sci.*, 169(2):161–184, 1996.

[10] Pavel Goralcik, A. Goralciková, and Václav Koubek. Alternation with a pebble. *Inf. Process. Lett.*, 38(1):7–13, 1991.

[11] F.C. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8:553–578, 1965.

[12] D. E. Knuth, J. H. Morris, and W. Rytter. Fast pattern matching in strings. *SIAM Journal of Computing*, 6:322–350, 1977.

[13] Pascal Michel. An NP-complete language accepted in linear time by a one-tape Turing machine. *Theor. Comput. Sci.*, 85(1):205–212, 1991.

[14] Torben Æ. Mogensen. WORM-2DPDAs: An extension to 2DPDAs that can be simulated in linear time. *Information Processing Letters*, 52:15–22, 1994.

[15] Burkhard Monien. Transformational methods and their application to complexity problems. *Acta Informatica*, 6:95–108, 1975.

[16] Benedek Nagy. The languages of SAT and n-SAT over finitely many variables are regular. *Bulletin of the EATCS*, 82:286–297, 2004.

[17] Thomas W. Reps. "Maximal-munch" tokenization in linear time. *ACM Trans. Program. Lang. Syst.*, 20(2):259–273, 1998.