Journal of Automata, Languages and Combinatorics **12** (2007) 1/2, 97–115
© Otto-von-Guericke-Universität Magdeburg

TWO-WAY FINITE AUTOMATA WITH A WRITE-ONCE TRACK [1]

Berke Durak
L.I.A.F.A., Université Paris VII,
2 place Jussieu, 75221 Paris, France
e-mail: berke@ouvaton.org

ABSTRACT

The basic finite automata model has been extended over the years with different acceptance modes (nondeterminism, alternation), new or improved devices (two-way heads, pebbles, nested pebbles) and with cooperation. None of these additions permits recognition of non-regular languages. The purpose of this work is to investigate a new kind of automata which is inspired by an extension of 2DPDAs. Mogensen enhanced these with what he called a WORM (write once, read many) track and showed that Cook's linear-time simulation result still holds. Here we trade the pushdown store for nondeterminism or a pebble and show that the languages of these new types of finite automata are still regular. The conjunction of alternation, or of nondeterminism and a pebble permits the recognition of non-regular languages. We give examples of languages that are easy to recognize and of operations that are easy to perform using these WORM tracks under nondeterminism. While somewhat similar to Hennie machines, our models do not require an explicit time bound on their computations.

Keywords: Automata models, WORM tracks, regularity

1. Introduction

Two-way deterministic pushdown automata (2DPDAs) have played an important role in the development of formal language theory [6]. It is a well-known fact [4, 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 closely related to the number of heads. By Cook's result [5], a k-head 2DPDA can be simulated on a random-access machine with unit cost 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" [7].

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. WORM

[1]Full version of a submission presented at the 7th Workshop on *Descriptional Complexity of Formal Systems* (Como, Italy, June 30 – July 2, 2005).

tracks are a subtle restriction of Turing machine tracks that preserves memoization, or caching, of the automata configurations. Memoization is the working principle of Cook's initial off-line 2DPDA simulation algorithm and the subsequent on-line variant due to [1]. The latter can, with almost no modification, simulate WORM machines in linear time.

As with ordinary Turing tapes, the squares of a WORM track are initially blank. However the automata-observable content of each square must only be a function of position. This means that reading a square must always return the same result, which implies two conditions. The first is that each square can be written at most once; writes to a non-blank cell should have no effect. The second is that a blank cell that has been read must always remain blank. Mogensen ensures this by automatically writing a special symbol θ to a blank cell before the automaton reads it. Although we initially used that approach, halting the computation upon read access to a blank square is equivalent (just write θ before each read) and shortens some proofs. Hence each square has only one automata-observable value.

WORM tracks thus serve as an auxiliary storage device permitting, for instance, lexical tokenization in linear time [17], as opposed to the quadratic worst-case time of current lexical scanners. WORM tracks are also useful for recognizing some languages more easily, such as $\{uu^Rvv^R \mid u, v \in \{a, b\}^*\}$ (PALSQUARE). However it is still an open question whether or not WORM-2DPDAs recognize more languages than 2DP-DAs. 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 nondeterminism or a pebble (thus obtaining what we call WORM-2NFAs and P-WORM-2DFAs), 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, nondeterministic 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.

2. WORM-2NFAs

WORM-2NFAs are somewhat similar to nondeterministic 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 [2]. 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 nondeterministic Turing machines recognizing non-regular, NP-complete languages[2] [13]. Furthermore, the linearity of running time is a non-trivial and thus undecidable property of Turing machines, making the class

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

of Hennie machines non-constructive. Hence WORM-2NFAs constitute an interesting class of nondeterministic machines with a writable tape and unrestricted running time and whose languages are regular. They may in fact be more succinct machines since they can solve SAT using a polynomial number of states.

2.1. 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 read/write head. The cells of the WORM tracks are called WORM cells or WORM squares. The principle behind WORM squares is that reading one must always return the same symbol through the computation. They are initially blank (we denote blank squares by $\Box$), that is, they have no initial value. There is only one way a WORM square can get its value, and that is when the automaton decides to write a symbol $\gamma \in \Gamma$ into it while it is a blank. Its content then becomes γ and stays that way; further writes will be silently be ignored – in particular, they will not cause the computation to abort or otherwise change, as this would be akin to returning a different value. Reading a blank square is not allowed and halts the computation.

The automaton can never observe blank squares without halting. A WORM track can thus be seen as a cache for memorizing intermediate results as in dynamic programming: its behaviour ensures that any two reads from the same square will return the same result.

We now give a formal definition of our model [3].

Definition 1 (k-WORM-2NFA) *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 is $\Gamma \cup \{\Box\}$. The set of transitions is T. 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. 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 as given by τ.

Definition 2 (Transitions) *Let $t \in T$ be a transition. We define $\alpha(t) = q \in Q$ to be the source and $\omega(t) = q' \in Q$ to be the target states of t.*

- *If $\tau(q) = C$, the transition is of the form $t = (q, \sigma, d, q')$ 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).*

[3] Unless specified otherwise, alphabets are finite, non-empty sets. We write X^* for the set of words over an alphabet X, and X^n for the subset of X^* of words of length n (for $n \geq 0$). The empty word is ε and the length of a word u is written $|u|$, while its i-th letter is written u_i ($1 \leq i \leq |u|$).

- *When $\tau(q) = R$ or $\tau(q) = W$, the transition is of the form $t = (q, \gamma, q')$, where $\gamma \in \Gamma$ gives a letter to read or write from one of the WORM tracks.*

For simplifying analysis, we want the automaton to finish its computations with its head on the right end of the input. Thus we require the set T to contain all transitions of the form $(q^+, \sigma, 1, q^+)$ (for $\sigma \in \Sigma$) and no transitions of the form (q^+, σ, d, q) with $q \neq q^+$.

Definition 3 (Configurations) *A* configuration *of M over an input $u \in \Sigma^*$ of length m is a triple (q, i, w) where $q \in Q$ is the state, $1 \leq i \leq m$ is the position of the head and $w = (w_1, \dots, w_k)$ are the WORM contents with each $w_i \in (\Gamma \cup \{\Box\})^m$. Note that $\Box \notin \Gamma$. The* initial *configuration is $(q^0, 1, (\Box^m, \dots, \Box^m))$. 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 4 (Effects of transitions) *Let t be a transition and $c = (q, i, w)$ 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 $1 \leq i + d \leq m$ holds, the transition is applicable and 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 that square, we have two cases:*
  - *If the square is not blank and contains the symbol γ, the transition is applicable and $c \cdot t = (q', i, w)$.*
  - *If the square is blank or contains a symbol other than γ, the transition is not applicable.*
- *When $t = (q, \gamma, q')$ with $\tau(q) = W$, γ is written to the i-th square of the $\kappa(q)$-th WORM track if that square 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 5 (k-WORM-DFA) *A k-WORM-2NFA is deterministic and called a k-WORM-DFA when there is at most a unique transition that can apply to any given state.*

Definition 6 (Runs, computations, language) *A* run *is a finite sequence $t_1 t_2 \dots t_n$ of transitions. A* computation *(over the input u) is a run $t_1 \dots t_n$ that can be applied to the initial configuration $c^0 = (q^0, 1, (\Box^m)^k)$, 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 computation then leads from c^0 to c. Such a computation is* accepting *when the obtained configuration is final, that is, when $q = q^+$. A run $t_1 \dots t_n$ is said to* lead *from a configuration c to c' if it can be applied to c to give c'. 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: $\alpha(t_1 \dots t_n) = \alpha(t_1)$ and $\omega(t_1 \dots t_n) = \omega(t_n)$ when $n \geq 1$. Note that $\omega(t_i) = \alpha(t_{i+1})$ for $1 \leq i < n$. We now give an example showing the usefulness of WORM tracks.

Example 1 In [3], given a language L over an alphabet Σ, the set Collage(L) was defined as the set of words that can be obtained by pasting arbitrary words chosen from L one on top of the other at random positions.

More precisely, a word w is said to be a collage of words from L when it can be obtained as the limit of a sequence $u_0, \dots$ defined as follows.

- Let $\Box \notin \Sigma$ be a distinct symbol.
- Let $n = |w|$ and let $u_0 = \Box^n \in (\Sigma \cup \{\Box\})^*$ be the word of length n containing only the symbol $\Box$.
- If $u_i \in \Sigma^*$, that is, if u has no occurences of $\Box$, then $u_{i+1} = u_i$.
- Otherwise, select $v_i \in L$ such that $|v| \leq n$ and j_i such that $1 \leq j_i \leq n - |v_i| + 1$. Then u_{i+1} is the word u_i where the subword at positions j_i to $j_i + |v_i| - 1$ has been replaced by v_i.

As an example let $L = \{a, aa, aab, baba, aaabbb, abaabb\}$. Then the word $w = abaababaabbbb$ is in Collage(L) as can be seen in Figure 1(a). Actually, this collage idea was inspired by WORM tracks.

It was shown in [3] that if L is regular, then Collage(L) is also regular. This result is now a consequence of Theorem 1. Indeed let L be a regular language recognized by a one-way nondeterministic automaton M having n states. It is very easy to give an $O(n)$-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). Hence, by Theorem 1, Collage(L) is regular,

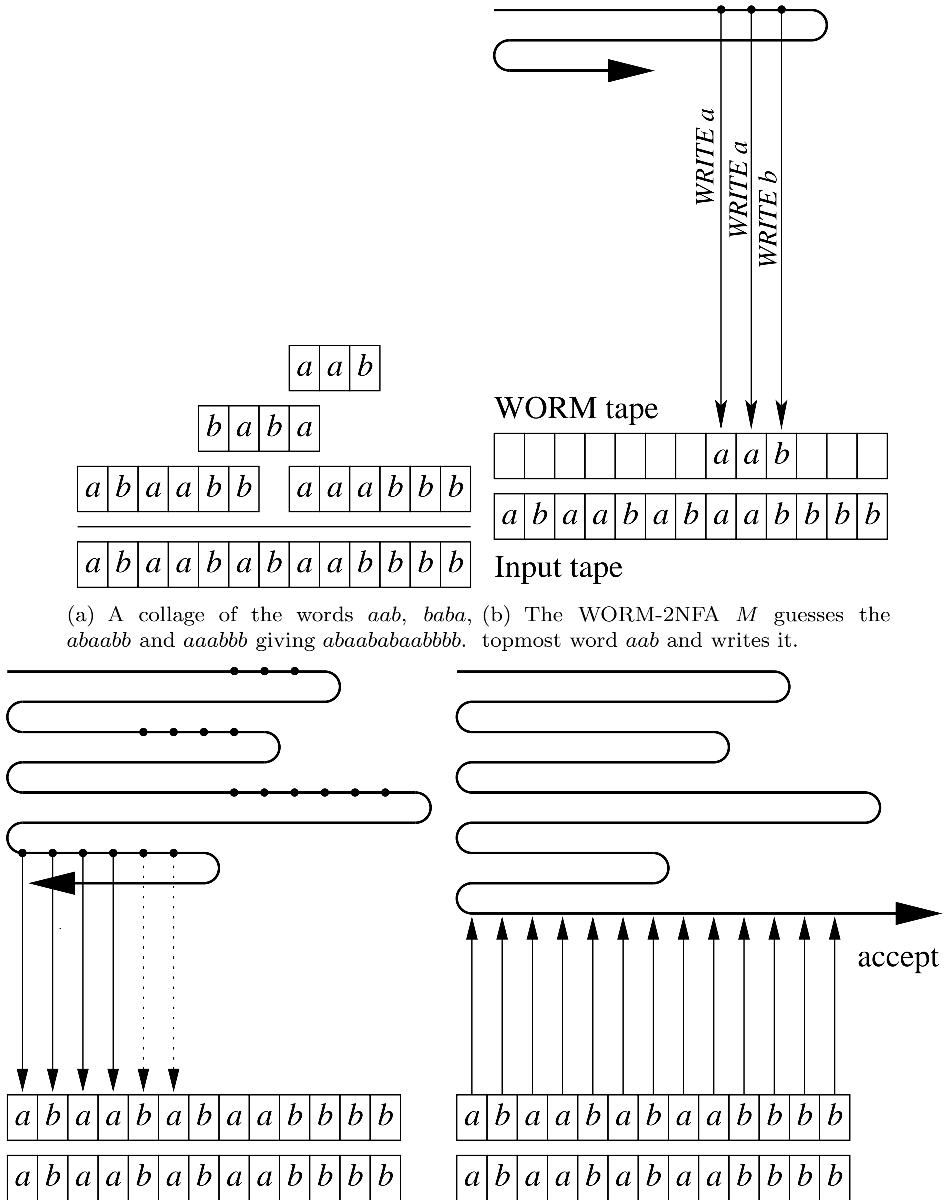
2.2. The Membership Problem

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. Nondeterministic 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.

A WORM-2DFA, being deterministic, cannot be twice at the same position and in the same state without looping. Hence, for a given WORM-2DFA, accepting computations have a length linear in the size of the input, which makes their languages regular by [11]. This does not hold for WORM-2NFAs.



(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)

Proposition 1 *There exists a WORM-2NFA whose accepting computations can have a length quadratic in their input.*

Proof. Consider inputs over the alphabet $\{a, b\}$ of the form ab^*a. Let M be a 1-WORM-2NFA repeating the following operation an arbitrary number of times.

(1) Place the head on an arbitrary position.
(2) Write X on the WORM track.
(3) Move the head to the beginning of the input (which is the first a).
(4) Move the head to the end of the input (which is the last a).

When M nondeterministically decides to stop performing the above steps, it checks that all WORM cells have been written with an X. As M must tick m cells, where m is the length of its input, and as between steps (1)–(4) it must perform $\geq m$ transitions, it follows that an accepting computation of M has a length in $\Theta(m^2)$. □

This quadratic bound is tight.

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

Proof. As M behaves as a 2NFA between two modifications of the WORM tracks, and since there cannot be more modifications than WORM squares, a computation of M can be decomposed into $\leq k|u|$ segments delimited by modifications of the WORM tracks It is easily shown that every computation of a 2NFA is equivalent to a computation of length less than the product of the number of states and the length of the input. It can therefore be assumed that the segments have length $\leq n|u|$. The total length of the computation is then $\leq nk|u|^2$. □

Using an idea similar to [16] or to [2], we can give, for every $n \geq 1$, a WORM-2NFA of size polynomial in n and accepting the positive instances of n gates or less of the circuit satisfiability (CIRCUIT-SAT) problem. For that, 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 $\#x_iI$ (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, using a number of states polynomial in n.

Lemma 2 *For every n there is an $O(n^2)$-state 1-WORM-2NFA M_n accepting S_n^+, whose description can be computed in time polynomial in n.*

Proof. Let $u \in \Sigma_n^*$ be the input word. The machine M_n first checks that u is syntactically correct. Then, it nondeterministically guesses a valuation for the circuit by writing 1 or 0 under the defining occurrence of every input gate. After that, it enters a loop where it selects an unvisited internal gate, and reads its type t (which can be AND, OR or NOT) its number i as well as the number j of its first preceding gate. These two numbers are stored in its finite control, thus requiring $O(n^2)$ states. The machine M then scans its input, looking for the defining occurrence of the gate j, and reads on its WORM track the boolean value that has been computed (if j is an internal gate) or guessed (if j is an input gate) for it. (If this value has not yet been defined, M reads an empty square and thus halts, having badly chosen its visiting order.) This boolean value is stored as one bit in the control state. Then, M goes back to the defining occurrence of gate i, and if $t = \text{AND}$ or $t = \text{OR}$, reads the number k of the second input gate of gate i and collects, in the same manner, the value computed or guessed for that gate. Then M combines, according to the boolean operation specified by t, the bits thus read, and writes the result on the WORM cell under the defining occurrence of gate i. This procedure goes on until M computes the value of gate 1, and accepts if it is true. □

In fact, the machines M_n given in the above proof can be transformed into two-way deterministic automata with a length-preserving homomorphism [2] by making them guess the values of all gates, including inputs, before checking their consistency; then, once the values have been guessed, the WORM track is not written to any more.

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

Proof. The language $\text{CIRCUIT} - \text{SAT} = S^+ = \bigcup_{n\geq 1} S_n^+$ is NP-complete [8]. (To be precise, it is necessary to reencode S^+ over a finite alphabet – this will induce a logarithmic expansion factor.) Let $\langle M\rangle\#u$ 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 $\geq km|u|^2$ transitions ending in the accepting state. Such a list can be guessed by a nondeterministic 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 converted by Lemma 2 into an $O(n^2)$-sized instance of the membership problem, in polynomial time. This shows NP-hardness. □

2.3. 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 [9].) 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[4] and hence is regular.

Definition 7 (Splicings) *Let $c = t_1 \dots t_n$ be a computation. The* splicing *of c at position p ($1 \leq p < m$, where m is the length of the input) is a factorization $c = a_1 a_2 \dots a_r$ of c into runs $a_1, \dots, a_r$ 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 $> p$).*

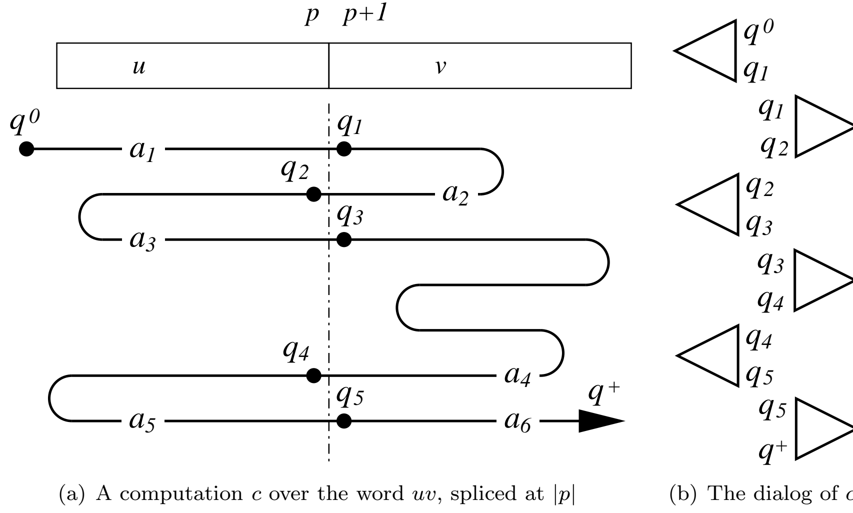
Our requirement of inclusion for transitions of the form $(q^+, \sigma, 1, q^+)$ ensures that we can restrict ourselves to accepting computations which finish by having the head scanning the remaining of the input to the right in state q^+: any splicing of such a computation will finish in the right half and thus have an even number of blocks.

Definition 8 (Triangles, dialogs) *Given a computation c spliced at p as $c = a_1 a_2 \dots a_r$, we look at the exchange of states between the left and right halves (i. e., between odd- and even-numbered runs). For odd j the automaton leaves the left half 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 $\tilde{a}_{j+1} = [q_1 \triangleright q_2]$ which we call a (left)* triangle*. Similarly, for even j, we 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: $\mathrm{dlg}_p(c) = \tilde{a}_1 \tilde{a}_2 \dots \tilde{a}_r$.*

In Figure 2(a) we have a computation c over a word uv spliced at position $|u| = p$ as $c = a_1 a_2 \dots a_6$. 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)):

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

[4]The left residual of a language L by a word u is the set $u^{-1}.L = \{v \mid uv \in L\}$ which is is the language a one-way automata accepting L accepts after having read u



(a) A computation c over the word uv, spliced at $|p|$ (b) The dialog of c

Figure 2: The dialog of a computation

Take two accepting computations c and c' over uv and wx respectively. It has already been noted in [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 $c = a_1 \dots a_r$ and $c' = b_1 \dots b_r$ be the splicings of c and c' at $|u|$ and $|w|$ respectively. Then $c'' = a_1 b_2 a_3 b_4 \dots b_r$ 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 the computation goes on, WORM squares get filled by write operations. At the end of the computation the j-th WORM track contains the word $w_j \in (\Gamma \cup \{\Box\})^m$ 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. However the computation c will still be possible. Indeed, for every WORM square, look at the first access to that square in c. As the computation c was successful, this access must be a write. Hence M will try, in the second run of c over the already filled WORM tracks, to write the same symbol again to that square. This will pose no problem as rewriting is allowed. Further accesses happen in the same way. Generalizing this observation, we obtain the following Lemma:

Lemma 3 (Repeatability) *Let M be a WORM-2NFA over an input u. If $a = t_1 \dots t_n$ is a run 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 $c = a_1 \dots a_k$ and $c' = b_1 \dots b_\ell$. 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 9 *Let X be a possibly infinite alphabet; X^* stands for the set of finite sequences over X. We recursively define the operation $I : X^* \to X^*$ as follows:*

$$I(\varepsilon) = \varepsilon$$

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

The sequence $I(x)$ is called the novelty sequence *of x.*

This operation keeps only the first occurrence of each symbol, deleting repeated symbols. Thus if $X = \{0, 1, 2, \dots, 9\}$ we have $I(314159265358) = 31459268$.

We can now give the core lemma of our result.

Lemma 4 *Let c and c' be two accepting computations over uv and wx respectively. Let $D = I(\mathrm{dlg}_{|u|}(c))$ and $D' = I(\mathrm{dlg}_{|w|}(c'))$ be the respective novelty sequences of their dialogs spliced at $|u|$ and $|w|$. If $D = D'$ then there exists an accepting computation c'' over ux.*

Proof. Let $c = a_1 \dots a_k$ and $c' = b_1 \dots b_\ell$ be the 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 prerequisites 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.

Say that a run x is *novel* (or a *novelty*) in a splicing of 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 $N \subseteq \{1, \dots, k\}$ (resp. $N' \subseteq \{1, \dots, \ell\}$) 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 $\lambda_1 < \dots < \lambda_n$ and $\lambda'_1 < \dots < \lambda'_n$ 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 $a_{\lambda_{\nu-1}+1}, \ldots, a_{\lambda_\nu}$ and $b_{\lambda'_{\nu-1}+1}, \ldots, b_{\lambda'_\nu}$.

Say that 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 $d_1 \ldots d_m$ be the splicing of c'' at position $|u|$. Let X be the 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 over X consisting of the left runs of d. We can then use the operation I over X^* as in Definition 9 to give the condition:

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

This means that, modulo repetitions, c'' consists 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 necessarily novel, i. e., $\lambda_1 = \lambda'_1 = 1$ and $\lambda_2 = \lambda'_2 = 2$. Since $D = D'$, $c'' = a_1$ and $c'' = a_1 b_2$ 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 $a = a_1 \ldots a_{\lambda_\nu} L a_{\lambda_{\nu+1}} \ldots a_k$ and $b = b_1 \ldots b_{\lambda'_\nu} R b_{\lambda'_{\nu+1}} \ldots b_\ell$.

For c'' to satisfy equation (1) for ν, the left runs of L and the right runs of R must be executed in order (as they are prerequisites of the computation) and then $b_{\lambda'_{\nu+1}}$ must be added. 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 induction hypothesis, each left run of R has the same triangle as some left run of c''_0. By the repeatability lemma (Lemma 3), 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 sequence of runs R' over ux whose corresponding computation 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 the symmetric modification of 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 subcomputation P of c''_0 leading from q_2 in the right half to q_1 in the corresponding half. As this subcomputation 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 construct the symmetric modification of 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. Now note that the computation c'' over ux is accepting, since the construction preserves the sequence of novelties and thus has a triangle involving the final state q^+. □

As an example consider the computations whose dialogs are:

$$c : \begin{bmatrix} & [0 \triangleleft 1] \\ [1 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 2] & [2 \triangleleft 4] \\ [4 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 2] & [2 \triangleleft 4] \\ [4 \triangleright 2] & [2 \triangleleft 1] \\ [1 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 5] & \end{bmatrix} \quad c' : \begin{bmatrix} & [0 \triangleleft 1] \\ [1 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 2] & [2 \triangleleft 4] \\ [4 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 2] & [2 \triangleleft 4] \\ [4 \triangleright 2] & [2 \triangleleft 1] \\ [1 \triangleright 2] & [2 \triangleleft 4] \\ [4 \triangleright 2] & [2 \triangleleft 3] \\ [3 \triangleright 5] & \end{bmatrix}$$

Our construction gives a computation whose dialog is:

$$c'' : \begin{bmatrix} & [0 \triangleright 1] \\ [1 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 2] & [2 \triangleleft 4] \\ [4 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 2] & [2 \triangleleft 4] \\ [4 \triangleleft 2] & [2 \triangleleft 1] \\ [1 \triangleleft 2] & [2 \triangleleft 4] \\ [4 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 2] & [2 \triangleleft 4] \\ [4 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 2] & [2 \triangleleft 3] \\ [3 \triangleleft 5] & \end{bmatrix}$$

We are now ready to prove our main result.

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

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

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

We show that if $F_u = F_w$ for two words u and w then $u^{-1}.\mathscr{L}(M) = v^{-1}.\mathscr{L}(M)$, i. e., for all x, we have $ux \in \mathscr{L}(M)$ if and only if $wx \in \mathscr{L}(M)$.

By symmetry, it suffices to show one implication of the equivalence. Suppose there exists an accepting computation c' over wx. Then the sequence $D = I(\mathrm{dlg}_{|w|}(c'))$ is

in F_u and hence in F_w. Therefore there is an accepting computation c over uv such that $I(\mathrm{dlg}_{|u|}(c)) = D$, where v is some word. By Lemma 4 there is an accepting computation over 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 determined 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. □

Note that the proof is constructive since the set F_u can be computed by enumerating novelty sequences and testing them individually. Accepting computations can be enumerated with the help of the quadratic bound on computation lengths, and the regularity result which limits the length of the words we must consider. (The number of residuals and hence the number of states of a deterministic automaton accepting the same language as M is bounded by a the number of possible sets of novelty sequences.)

2.4. Conciseness

A WORM-2NFA can easily recognize the image by a length preserving homomorphism of a 2NFA language by guessing a pre-image on its WORM track and then executing the 2DFA or 2NFA. It has been shown in [2], Theorem 4.1 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 succinct than 2NFAs, it follows that WORM-2NFAs are at least exponentially more succinct than 2NFAs.

3. Further Extensions

3.1. Stringent Cells

The regularity of WORM-2NFAs is very sensitive to the functionality of the WORM cells. Consider a variant of WORM cells where any attempt to write to a non-blank cell causes the computation to halt. Call these *stringent* WORM cells.

Proposition 3 *Nonregular languages can be accepted by WORM-2NFAs with stringent cells.*

Proof. Let $\Sigma = \{\triangleleft, \triangleright, a, b\}$. The language $L = \{\triangleleft a^p b^p \triangleright u \mid p \geq 0, u \in \Sigma^*\}$ can be accepted by a 1-WORM-2NFA endowed with stringent WORM cells. The automaton first checks that the input is of the form $\triangleleft a^* b^* \triangleright \Sigma^*$. It then enters a nondeterministic loop where it selects a position under an a in the left half and marks the square with an X, switches to the second half and does the same under a b. After nondeterministically exiting the loop, the automaton checks that all positions between the first $\triangleleft$ and $\triangleright$ have been checked and accepts. □

3.2. Alternation

It is possible to define alternating WORM machines in the same way as alternating Turing machines are defined. This gives WORM-2AFAs. Each subcomputation of a WORM-2AFA gets a copy of the WORM track which can be modified independently of the others. The nodes of the alternating computation tree are labelled with the contents of the WORM tracks. However this makes simulation of stringent cells possible.

Proposition 4 *For every WORM-2NFA M running with stringent WORM cells, there exists a WORM-2AFA M' running with ordinary WORM cells and accepting the same language as M.*

Proof. As each branch of a universal fork has its own copy of the WORM track, one of these branches can be used to check if a given cell is blank without affecting the same cell in the other branches. This way M can be modified into a WORM-2AFA M' by predicating each write operation with a blankness check, such that the input is rejected if the check fails. □

3.3. Pebbles

Since alternation gets us out of the realm of regular languages, we are confined to nondeterministic acceptance modes. 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 [2, 10, 9]. It is then natural to extend WORM automata with a pebble. This gives P-WORM-2DFAs and P-WORM-2NFAs. However, adding a single pebble to WORM-2NFAs permits recognition of non-regular languages. As P-WORM-2DFAs can have computations of quadratic length, the result of [11] does not apply. Actually, pebbles and WORM tracks give regular languages only under determinism.

Theorem 2 *P-WORM-2DFAs have regular languages.*

Proof. Let M be a P-WORM-2DFA having Q as its set of states, with $q^+ \in Q$ as its only accepting state. We assume that the model is suitably enhanced with endmarkers to allow M to recognize the beginning and end of its input. The automaton M can reject its input by blocking or by looping in a non-accepting state. Without loss of generality, we further assume that M moves its head to the right with the pebble end before accepting. We will refer to this property by saying that M is *right-finishing.* Let $uv \in \mathscr{L}(M)$ be an accepted input of length n, where u is its left half and v its right half. For brevity, we omit a complete formalization of P-WORM-2DFAs. However a configuration of M can be represented by a quadruple (q, i, j, w) where

- $q \in Q$ is the control state,
- i is the head position ($1 \leq i \leq n+2$, with the beginning and end markers at $i = 1$ and $i = n+2$ respectively),

- j gives the pebble position ($0 \leq j \neq n+2$), $j > 0$ meaning that the pebble is dropped at j, and $j = 0$ meaning that the pebble is picked by the head,
- $w \in \left(\Gamma \cup \{\square\}\right)^{n+2}$ gives the contents of the WORM track.

The initial configuration is $(q_0, 2, 0, \square^{n+2})$. Let $c = (q_r, i_r, j_r, w_r)_{r \geq 0}$ be the sequence of configurations of M in its computation over uv, naming initial configuration as (q_0, i_0, j_0, w_0). In the r-th configuration ($r \geq 0$) we say that the pebble is in the left if the pebble is picked and the head is in the left half ($j_r = 0$ and $i_r \leq |u| + 1$), or if the pebble is dropped in the left half ($j_r \leq |u| + 1$); otherwise, the pebble is said to be in the right half. Factorize the computation c into episodes as $c = e_0 e_1 \ldots e_p$ such that for all even (resp. odd) s, the pebble is in the left (resp. in the right) in episode e_s. Hence, at the beginning of each episode, the pebble is taken and the head is in the same position (which is $|u|$ or $|u| + 1$ depending on the half in which the episode takes place). Because M is right-finishing, we have $p \geq 1$ and p odd. For every s ($1 \leq s \leq r$), let $\alpha(e_s)$ stand for the control state of the first configuration of episode e_s (thus $\alpha(e_0) = q_0$). As Q is finite, there exist p and r such that $0 \leq p < r \leq \mathrm{Card}(Q)$ such that $\alpha(e_p) = \alpha(e_r)$. Observe that during each episode, the half that does not contain the pebble is reduced to a WORM-2DFA.

For every p ($1 \leq p \leq r$) let h_p ($h_p \in (Q \times Q)^*$) be the pebble crossing sequence of the episode p, that is, the sequence of couples $(q_1, q'_1) \ldots (q_k, q'_k)$ where q_ℓ is the state by which the half that does not contain the pebble is called, and q'_ℓ is the state by which it answers. Note that, since the automaton accepts its input, it does not loop, and h_p has finite length. Moreover, by determinism, states in h_p cannot repeat, and thus the set of possible such crossing sequences is finite for a fixed set of states Q.

Let I be the operation of definition 9 suppressing repeated entries of a sequence.

Define the sequence

$$\begin{aligned} S_{u,v} = &(\alpha(e_0), I(h_0)) \cdot \\ &(\alpha(e_1), I(h_0 h_1)) \cdot \\ &\ldots \\ &(\alpha(e_r), I(h_0 h_1 \cdots h_r)) \end{aligned}$$

The set $\mathscr{S} = \{S_{u,v} \mid u, v \in \Sigma^*\}$ of these sequences over various entries of the form uv is finite. A left residual $u^{-1}.\mathscr{L}(M)$ can be described by a subset

$$G_u = \left\{ S_{u,v} \;\middle|\; v \in \Sigma^* \right\}$$

of $\mathscr{S}$. Therefore $\mathscr{L}(M)$ has a finite number of left residuals and is regular. $\square$

Figure 3 illustrates the exchange of pebble and control states between the two halves. The computation consists there of three episodes e_0, e_1, e_2. The automaton starts in state q_0 in episode e_0, during which the right half is called in state q_1 and answers in state q_2; then is called again in state q_3 and answers in state q_4. The episode e_0 then ends as the right half is called in state q_5 with the pebble. Thus $h_0 = (q_1, q_2) \cdot (q_3, q_4)$. Similarly $h_1 = (q_6, q_7) \cdot (q_8, q_9)$ and $h_2 = (q_{11}, q_{12}) \cdot (q_{13}, q_{14})$. The sequence $S_{u,v}$ is

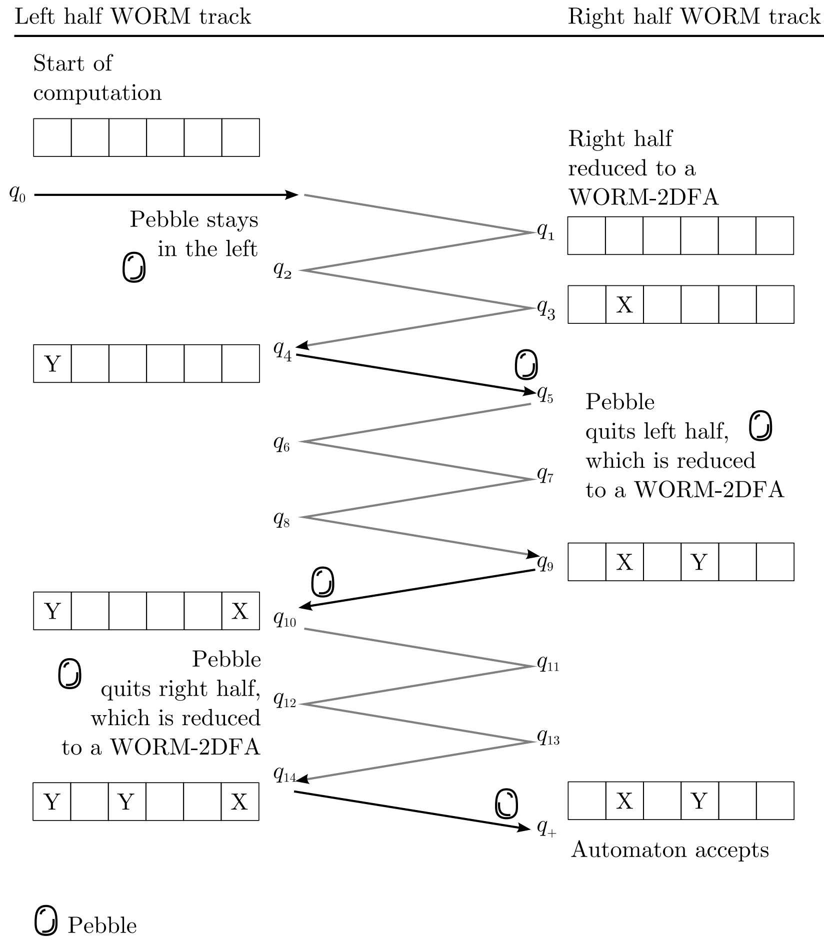


Figure 3: Pebble and control state exchanges between the two halves of the input in a P-WORM-2DFA, showing the evolution of the WORM track contents

then

$$\Big(q_0,\quad I\big((q_1,q_2)\cdot(q_3,q_4)\big)\Big)\cdot$$
$$\Big(q_5,\quad I\big((q_1,q_2)\cdot(q_3,q_4)\cdot(q_6,q_7)\cdot(q_8,q_9)\big)\Big)\cdot$$
$$\Big(q_{10},\quad I\big((q_1,q_2)\cdot(q_3,q_4)\cdot(q_6,q_7)\cdot(q_8,q_9)\big)\Big)\cdot(q_{11},q_{12})\cdot(q_{13},q_{14})\big)\Big)$$

Proposition 5 *P-WORM-2NFAs can accept some non-regular languages.*

Proof. The language $\left\{a^{p+q}b^p \mid p, q \geq 0\right\}$ can be accepted by a P-WORM-2NFA M as follows. First, M checks that the input word is in a^*b^*. Then, M drops its pebble on the first letter of its input, and repeats the following procedure:

(1) Tick the WORM cell under the pebble.

(2) Nondeterministically select a position whose input letter is a b.

(3) Tick the WORM cell under that position.

(4) Go back to the pebble.

(5) Pick the pebble, advance it one position to the right and drop it.

(6) If the pebble is still over an a, go to (1).

After that, M checks that all WORM cells have been ticked and accepts its input. Thus, for every a, M checks at most one b. □

4. Concluding Remarks and Open Questions

The ability of WORM-2NFAs 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 nondeterministic 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 nondeterministic 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.

The effect of WORM cells on the languages of other kinds of finite computing devices, such as 2DFAs with nested pebbles, 2DFAs or 2NFAs with monotonic output tapes and tree-walking automata remains to be explored.

Acknowledgements

I would like to thank Prof. Ch. Choffrut for his helpful advice and the many lengthy discussions about this subject, and an anonymous referee for very extensive comments, corrections and suggestions.

References

[1] N. ANDERSEN, N. D. JONES, Generalizing cook's transformation to imperative stack programs. *Proceedings of the Colloquium in Honor of Arto Salomaa on Results and Trends in Theoretical Computer Science* ISBN 3-540-58131-6, Springer-Verlag, London, UK, 1994, 1–18.

[2] J.-C. BIRGET, Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory* **29** (1996) 3, 191–226.

[3] C. CHOFFRUT, B. DURAK, Collage of two-dimensional words. *Theoret. Comp. Sci* **340** (2005) 1, 364–380.

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

[5] S. A. Cook, Linear-time simulation of deterministic two-way pushdown automata. *Information Processing* **71** (1972), 75–80.

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

[7] Z. Galil, J. Seiferas, A linear-time on-line recognition algorithm for "Palstar". *Journal of the Association for Computing Machinery* **25** (1978), 102–111.

[8] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

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

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

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

[12] D. E. Knuth, J. H. Morris, V. Pratt, Fast pattern matching in strings. *SIAM Journal of Computing* **6** (1977), 322–350.

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

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

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

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

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

(Received: October 23, 2005; revised: November 22, 2007)