RAIRO-Theor. Inf. Appl. 44 (2010) 507–523
DOI: 10.1051/ita/2011001

Available online at:
www.rairo-ita.org

TRANSLATION FROM CLASSICAL TWO-WAY AUTOMATA TO PEBBLE TWO-WAY AUTOMATA *

Viliam Geffert[1] and Ľubomíra Ištoňová[1]

Abstract. We study the relation between the standard two-way automata and more powerful devices, namely, two-way finite automata equipped with some ℓ additional "pebbles" that are movable along the input tape, but their use is restricted (nested) in a stack-like fashion. Similarly as in the case of the classical two-way machines, it is not known whether there exists a polynomial trade-off, in the number of states, between the nondeterministic and deterministic two-way automata with ℓ nested pebbles. However, we show that these two machine models are not independent: if there exists a polynomial trade-off for the classical two-way automata, then, for each $\ell \geq 0$, there must also exist a polynomial trade-off for the two-way automata with ℓ nested pebbles. Thus, we have an upward collapse (or a downward separation) from the classical two-way automata to more powerful pebble automata, still staying within the class of regular languages. The same upward collapse holds for complementation of nondeterministic two-way machines. These results are obtained by showing that each pebble machine can be, by using suitable inputs, simulated by a classical two-way automaton (and *vice versa*), with only a linear number of states, despite the existing exponential blow-up between the classical and pebble two-way machines.

Mathematics Subject Classification. 68Q45, 68Q70.

Keywords and phrases. Finite automata, regular languages, descriptional complexity.

* *Supported by the Slovak Grant Agency for Science (VEGA) under contract 1/0035/09 "Combinatorial Structures and Complexity of Algorithms".*

[1] Department of Computer Science, P. J. Šafárik University, Jesenná 5, 040 01 Košice, Slovakia;
viliam.geffert@upjs.sk; lubomira.istonova@upjs.sk

1. Introduction

The relation between determinism and nondeterminism is one of the key topics in theoretical computer science. The most famous one is the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question, but the oldest problem of this kind is $\mathrm{DSPACE}(n) \stackrel{?}{=} \mathrm{NSPACE}(n)$. Similarly, we do not know whether $\mathrm{DSPACE}(\log n) \stackrel{?}{=} \mathrm{NSPACE}(\log n)$. However, a positive answer for the $O(\log n)$ space would imply the positive answer for the $O(n)$ space, and hence the answers to these two questions are not independent. Analogically, a collapse of nondeterminism with determinism for the $O(\log \log n)$ space would imply the same collapse for the $O(\log n)$ space. (For a survey and bibliography about such translations, see *e.g.* [10,27].) Analogous upward translations can be derived for time complexity classes as well: $\mathrm{P} = \mathrm{NP}$ implies the collapse of nondeterminism with determinism for $2^{\Theta(n)}$ time.

At first glance, the problem has been resolved for finite state automata. Even a two-way nondeterministic finite automaton (2NFA, for short) and hence any simpler device as well (*e.g.*, its deterministic version, 2DFA) can recognize a regular language only. Thus, 2NFA's can be converted into deterministic one-way automata. However, the problem reappears, if we take into account the size of these automata, measured in the number of states.

On one hand, we know that eliminating nondeterminism in one-way n-state automata does not cost more than 2^n states (by the classical subset construction), and that there exist witness regular languages for which exactly 2^n states are indeed required. (For examples of such languages, see [18,20,23].)

On the other hand, we know very little about eliminating nondeterminism in the two-way case: it was conjectured by Sakoda and Sipser [22] that there must exist an exponential blow-up for the conversion of 2NFA's into 2DFA's. Nevertheless, the best known lower bound is $\Omega(n^2)$ [6], while the known conversion uses $2^{O(n^2)}$ states (converting actually into deterministic one-way machines) [21,24]. Thus, it is not clear whether there exists a polynomial trade-off. The problem has been attacked several times by proving exponential lower bounds for restricted versions of 2DFA's: Sipser [25] – for sweeping machines (changing the direction of the input head movement at the endmarkers only); Hromkovič and Schnitger [16] – for oblivious machines (moving the input head along the same trajectory on all inputs of the same length); Kapoutsis [17] – a computability separation for "moles" (seeing only a part of the input symbol thus traveling "in a network of tunnels" along the input). For machines accepting unary languages, a subexponential upper bound $2^{O(\log^2 n)}$ has been obtained [11].

It was even observed [22] that there exists a family of languages $\{B_n : n \geq 1\}$ which is *complete* for the two-way automata, playing the same role as, *e.g.*, the satisfiability of boolean formulas for the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question or the reachability in graphs for $\mathrm{DSPACE}(\log n) \stackrel{?}{=} \mathrm{NSPACE}(\log n)$: the trade-off between the 2NFA's and 2DFA's is polynomial if and only if it is polynomial for B_n, *i.e.*, if and only if B_n can be accepted by a 2DFA with a polynomial number of states. (For 2NFA's, n states are enough to accept B_n.)

In the absence of a solution for the general case, it is quite natural to ask whether some properties of the two-way automata cannot be translated into more powerful machines or language classes, in perfect analogy with the corresponding results for the upward translation established for the classical space and time complexity classes. So far, the only result of this kind [1] is that if an exponential trade-off between 2NFA's and 2DFA's could be obtained already by using a subset of the original language that consists of polynomially long strings, then DSPACE$(\log n) \neq$ NSPACE$(\log n)$.

In the same spirit, we shall study the relation between the standard two-way automata and more powerful devices, namely, two-way nondeterministic and deterministic finite automata equipped with a fixed number of additional "nested pebbles", movable along the input tape (PEBBLE$_\ell$-2NFA, PEBBLE$_\ell$-2DFA, respectively, with $\ell \geq 0$). If the ℓ pebbles could be moved around without restrictions, this would lead to nonregular languages. However, we consider another, more restricted, model from the literature, related to first-order logic with transitive closure. (See *e.g.* [8,13].) In the more restricted model, pebbles $\bullet_1, \ldots, \bullet_\ell$ are nested in a stack-like fashion: a level-i pebble $\bullet_i$ can be moved one position to the left or right *only if*, at the moment, no information is encoded by the use of the lower-level pebbles $\bullet_1, \ldots, \bullet_{i-1}$. (In our implementation, $\bullet_1, \ldots, \bullet_{i-1}$ must be placed on top of $\bullet_i$ each time $\bullet_i$ moves.) With this restriction, only regular languages are recognized [2,4,27]. However, measured in the number of states, the pebble machines are much more powerful. Even converting a PEBBLE$_1$-2DFA to a classical 2NFA may require an exponential blow-up, *i.e.*, the loss of the single pebble cannot be compensated economically by gaining nondeterminism. (See Thm. 2.1 below.)

Similarly as in the case of the classical two-way machines, we do not know whether there exists a polynomial trade-off between the PEBBLE$_1$-2NFA's and PEBBLE$_1$-2DFA's, the same problem reappears for PEBBLE$_\ell$-2NFA's and PEBBLE$_\ell$-2DFA's, with $\ell \geq 2$. However, we shall show that these machine models are related: if there exists a polynomial transformation from the classical 2NFA's to 2DFA's, then, for each $\ell \geq 1$, there must also exist a polynomial transformation, with the same degree of the polynomial, from the PEBBLE$_\ell$-2NFA's to PEBBLE$_\ell$-2DFA's. Thus, we have an upward collapse (and a downward separation) between the classical two-way automata and the much more powerful model using ℓ pebbles nested in the stack-like fashion, staying still within the class of regular languages.

A similar upward collapse holds for the trade-off between a two-way nondeterministic automaton accepting a language L and a machine for the complement of L: if the trade-off is polynomial for the classical 2NFA's, it must also be polynomial for the PEBBLE$_\ell$-2NFA's, for each $\ell \geq 1$. (So far, the problem is open for both these models.)

These results are obtained by showing that each PEBBLE$_\ell$-2NFA (or PEBBLE$_\ell$-2DFA) can be, by using suitable inputs, simulated by a classical 2NFA (or 2DFA, respectively) with only a linear number of states, despite the existing exponential blow-up between the classical and pebble machines. The same holds for the corresponding conversions in the opposite direction, from the classical machines to pebble machines.

2. Preliminaries

Here we introduce some basic notation and properties for a single-pebble computational model. For a more detailed exposition and bibliography related to regular languages, the reader is referred to [15,17,19].

A *two-way nondeterministic finite automaton* (2NFA, for short) is defined as a quintuple $A = (Q, \Sigma, \delta, q_{\text{I}}, F)$, in which Q is the finite set of states, Σ is the finite input alphabet, $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \rightarrow 2^{Q\times\{-1,0,+1\}}$ is the transition function, $\vdash, \dashv \notin \Sigma$ are two special symbols, called the left and the right endmarker, respectively, $q_{\text{I}} \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting (final) states.

The input is stored on the input tape surrounded by the two endmarkers. In one move, A reads an input symbol, changes its state, and moves the input head one cell to the right, left, or keeps it stationary, depending on whether δ returns $+1$, -1, or 0, respectively. The input head cannot move outside the zone delimited by the endmarkers: transitions in the form $\delta(q, \dashv) \ni (q', +1)$ or $\delta(q, \vdash) \ni (q', -1)$ are not allowed. If $|\delta(q, a)| > 1$, the machine makes *a nondeterministic choice*. If $|\delta(q, a)| = 0$, the machine *halts*.

The machine accepts the input, if there exists a computation path starting in the initial state q_{I} with the head on the left endmarker and reaching, anywhere along the input tape, an accepting state $q \in F$.

The automaton A is said to be *deterministic* (2DFA), whenever $|\delta(q, a)| \leq 1$, for all $q \in Q$ and $a \in \Sigma \cup \{\vdash, \dashv\}$.

We shall now introduce a more powerful model, namely, a two-way finite automaton equipped with a single additional "pebble" placed on the input tape. The action of the pebble machine depends on the current state, the currently scanned input tape symbol, and the presence of the pebble on this symbol. The action consists of changing the current state, moving the input head and, optionally, if the pebble is placed on the current symbol, moving also the pebble in the same direction.

Formally, a *one-pebble two-way nondeterministic finite automaton* (PEBBLE_1-2NFA, for short) is $A = (Q, \Sigma, \delta, q_{\text{I}}, F)$, where $Q, \Sigma, q_{\text{I}}, F$ are defined as above, but the transition function is of the form $\delta : Q \times (\Sigma \cup \Sigma^\bullet \cup \{\vdash, \dashv, \vdash^\bullet, \dashv^\bullet\}) \rightarrow 2^{Q\times\{-1,0,+1,-1^\bullet,+1^\bullet\}}$. The presence of the pebble on the current input tape symbol $a \in \Sigma \cup \{\vdash, \dashv\}$ is indicated by using the symbol $a^\bullet \in \Sigma^\bullet \cup \{\vdash^\bullet, \dashv^\bullet\}$, while the new input head movements $-1^\bullet, +1^\bullet$ are introduced to move the pebble. More precisely, a classical transition in the form $\delta(q, a) \ni (q', d)$, with $a \in \Sigma \cup \{\vdash, \dashv\}$ and $d \in \{-1, 0, +1\}$, is applicable only if the pebble is not placed on the current input tape symbol (change the current state from q to q' and move the input head in the direction d), while $\delta(q, a^\bullet) \ni (q', d)$ can be executed only if the pebble is placed on $a \in \Sigma \cup \{\vdash, \dashv\}$ at the moment (move the input head in the direction d, but leave the pebble in its original position). Finally, a transition $\delta(q, a^\bullet) \ni (q', d^\bullet)$, with $d^\bullet \in \{-1^\bullet, +1^\bullet\}$, moves also the pebble in the same direction d, together with the input head. Transitions in the form $\delta(q, a) \ni (q', d^\bullet)$ are meaningless, and hence not allowed.

The machine A starts its computation in the initial state q_{I} with both the input head and the pebble placed on the left endmarker, and accepts by reaching, anywhere along the input tape, a final state $q \in F$. Similarly, the final position of the pebble is irrelevant for acceptance.

A one-pebble two-way *deterministic* finite automaton (PEBBLE$_1$-2DFA) is defined in the usual way.

It is known [4] (see also Thm. 15.3.5 in [27]) that even nondeterministic Turing machines equipped with a single pebble and a worktape space of size $o(\log\log n)$ can accept regular languages only. Since PEBBLE$_1$-2NFA's may be viewed as one-pebble Turing machines with $O(1)$ worktape space, all models introduced above (2DFA, 2NFA, PEBBLE$_1$-2DFA, PEBBLE$_1$-2NFA) share the same expressive power – they all recognize the same class of regular languages.

However, if we take into account their number of states, the power is different. Converting a PEBBLE$_1$-2DFA to a classical 2NFA may require an exponential blow-up. That is, the loss of the pebble cannot be paid by gaining nondeterminism.

Theorem 2.1. *For each $m \geq 1$, there exists a finite unary language L_m that can be accepted by a* PEBBLE$_1$-2DFA *with $O(m^2 \log m)$ states, but for which each* 2NFA *requires at least $2^{\Omega(m \log m)}$ states.*

Proof. Let $M = p_1 \cdot p_2 \cdot \ldots \cdot p_m$, where p_i denotes the i-th prime, and let $L_m = \{1^\ell : \ell < M\}$.

The pebble machine A recognizing L_m utilizes the fact that $\ell < M$ if and only if no $x \in \{1, \ldots, \ell\}$ is a common multiple of $p_1, p_2, \ldots, p_m$. Therefore, A repeatedly checks, for $x = 1, \ldots, \ell$, if x is divisible by the primes $p_1, p_2, \ldots, p_m$. The value of x is represented by the distance of the pebble from the left endmarker. In order to check if p_i divides x, A traverses between the pebble position and the left endmarker and counts modulo p_i (alternating right-to-left traversals with left-to-right traversals for odd/even values of i). If A finds a prime p_i not dividing x, it does not check the next prime p_{i+1} but, rather, enters the initial state q_{I} in which it searches for the pebble and then moves the pebble one position to the right. After that, A can start checking the next value of x for divisibility by $p_1, p_2, \ldots, p_m$ or, if the pebble has reached the right endmarker, A can halt in an accepting state q_{F}. Carefully implemented, A uses only $2 + p_1 + p_2 + \ldots + p_m$ states. By the Prime Number Theorem (see, *e.g.*, [3,7]), we have $p_i = (1 + o(1)) \cdot i \cdot \ln i$, which gives $2 + p_1 + p_2 + \ldots + p_m \leq O(m^2 \log m)$.

On the other hand, each classical 2NFA A' recognizing L_m must use at least $M - 1$ states. This can be seen by the use of $n \to n + n!$ method [5,9,14]: on the input 1^{M-1}, a machine A' with fewer states than $M - 1$ cannot traverse the input tape from left to right without going into a loop, *i.e.*, without repeating the same state after traveling some h positions to the right, where $h \leq M - 1$. Thus, by iterating this loop $(M-1)!/h = \prod_{i=1, i \neq h}^{M-1} i$ more times, we get a valid computation path traversing $(M-1)+(M-1)!$ positions to the right. Therefore, if A' can get from a state q_1 to q_2 by traversing the entire input 1^{M-1}, it can also get from q_1 to q_2 by traversing the entire input $1^{(M-1)+(M-1)!}$. The same holds for right-to-left traversals and also for U-turns, *i.e.*, for computations starting and ending at the same endmarker. Thus, by induction on the number of visits at the endmarkers, we get that if A' accepts the input 1^{M-1}, it must also accept the input $1^{(M-1)+(M-1)!}$, which is a contradiction. Therefore, each 2NFA A' recognizing L_m must use at least $M-1 = p_1 \cdot p_2 \cdot \ldots \cdot p_m - 1$ states. Since $p_1 \cdot p_2 \cdot \ldots \cdot p_m \geq m^{\Omega(m)}$ (see, *e.g.*, Lem. 4.14 in [10]), we have $M-1 \geq 2^{\Omega(m \log m)}$. □

3. Translation to a single pebble

In this section, we first show that each PEBBLE$_1$-2NFA M (or PEBBLE$_1$-2DFA) can be, in a way, using a suitable encoding of the original input, simulated by a 2NFA M' (or 2DFA, respectively) without a pebble. Then we shall show the corresponding conversions in the opposite direction, from the classical two-way machines to two-way machines equipped with a pebble. The cost, in the number of states, will be linear for all these conversions, despite the exponential blow-up presented by Theorem 2.1. After that, we shall derive some consequences of these translations.

In what follows, we shall need a function P that maps each input w of the given pebble automaton M into a new word $P(w)$ providing all possible positions of the pebble in w. This image $P(w)$ can be used as an input for a classical automaton M' (no pebble), such that M' accepts $P(w)$ if and only if M accepts w. Let $P : \Sigma^* \rightarrow (\Sigma \cup \{\triangleright, \triangleleft\} \cup \Sigma^{\Box})^*$ map a word $w = a_1 \ldots a_k$ as follows:

$$\begin{aligned} P(a_1 \ldots a_k) \;\; = \;\; & a_1 \ldots a_k \triangleleft \triangleright a_1^{\Box} a_2 \ldots a_k \triangleleft \triangleright a_1 a_2^{\Box} \ldots a_k \triangleleft \triangleright \ldots \\ & \ldots \triangleleft \triangleright a_1 \ldots a_{k-1}^{\Box} a_k \triangleleft \triangleright a_1 \ldots a_{k-1} a_k^{\Box} \triangleleft \triangleright a_1 \ldots a_k \, , \end{aligned} \tag{3.1}$$

where $\triangleright, \triangleleft$ are new symbols and $\Sigma^{\Box} = \{a^{\Box} : a \in \Sigma\}$. That is, $\Sigma^{\Box}$ simply denotes the letters of the original alphabet marked by some box.

Thus, $P(a_1 \ldots a_k)$ consists of $k+2$ segments, enumerated from 0. The p-th segment, for $p = 0, \ldots, k+1$, will be used by M' to simulate M in situations when M has the pebble placed on the p-th position of the input tape. For these reasons, the p-th segment is of type $\triangleright a_1 \ldots a_p^{\Box} \ldots a_k \triangleleft$, that is, the p-th symbol is marked by the box. (Except for $p = 0$ and $p = k+1$, there is exactly one such "pseudo pebble" in each segment.) The symbols $\triangleright$ and $\triangleleft$ are the so called "stoppers", imitating the left and right endmarkers of the original input tape. The first and last segments are of special kind, representing the situations when M has the pebble placed on the left or right endmarker, respectively, with no letters marked by the box.

As an example, if $w = a_1 a_2 a_3$, then $P(a_1 a_2 a_3) = a_1 a_2 a_3 \triangleleft \triangleright a_1^{\Box} a_2 a_3 \triangleleft \triangleright a_1 a_2^{\Box} a_3 \triangleleft \triangleright a_1 a_2 a_3^{\Box} \triangleleft \triangleright a_1 a_2 a_3$. Thus, taking also into account the endmarkers, the input tape for the pebble automaton M is in the form $\vdash a_1 a_2 a_3 \dashv$ while the input tape for a classical automaton M' (no pebble) in the form $\vdash a_1 a_2 a_3 \triangleleft \triangleright a_1^{\Box} a_2 a_3 \triangleleft \triangleright a_1 a_2^{\Box} a_3 \triangleleft \triangleright a_1 a_2 a_3^{\Box} \triangleleft \triangleright a_1 a_2 a_3 \dashv$. Similarly, for $w = \varepsilon$, we have $P(\varepsilon) = \varepsilon \triangleleft \triangleright \varepsilon = \triangleleft \triangleright$, that is, the input tapes for M and M' are $\vdash\dashv$ and $\vdash\triangleleft\triangleright\dashv$, respectively. Therefore, the left and right endmarkers can be handled by M' as if marked by the "pseudo pebble" box, that is, the symbols $\vdash,\dashv$ can be viewed as if equal to $\vdash^{\square},\dashv^{\square}$, respectively.

Now we have all we need to prove the following theorem:

Theorem 3.1. (a) *For each* PEBBLE$_1$-2NFA $M=(Q,\Sigma,\delta,q_{\rm I},F)$ *with* m *states, there exists a classical* 2NFA $M'=(Q',\Sigma',\delta',q'_{\rm I},F')$ *with at most* $3\cdot m$ *states such that, for each input* $w\in\Sigma^*$*,* M' *accepts* $w'=P(w)\in\Sigma'^*$ *if and only if* M *accepts* w*. Here* $\Sigma'=\Sigma\cup\{\triangleright,\triangleleft\}\cup\Sigma^{\square}$ *and* P *denotes the mapping function defined by* (3.1)*.*

(b) *Moreover, if* M *is deterministic, then so is* M'*.*

Proof. Note that M' does not have to check whether its input $w'\in\Sigma'^*$ is indeed a valid image obtained by the use of the mapping P, *i.e.*, whether $w'=P(w)$, for some $w\in\Sigma^*$. Assuming that $w'=P(w)$, M' simply checks whether M accepts w. If this assumption is wrong, the answer of M' can be quite arbitrary. (In general, an input string $w'\in\Sigma'^*$ does not necessarily have the structure described by (3.1), for any $w\in\Sigma^*$.)

The basic idea is as follows. If, during the simulation, M has its pebble placed on the p-th position, M' works within the p-th segment of $P(w)$. The simulation is quite straightforward and M' does not have to leave this segment until the moment when M moves its pebble. Recall that M' relies on the assumption that the p-th segment contains one exact copy of w, correctly enclosed in between the symbols $\triangleright$ and $\triangleleft$, and that the current pebble position of M is clearly marked inside this segment, *i.e.*, there is exactly one symbol marked with the box, namely, the symbol on the p-th position. If this never-verified assumption were wrong, the simulation could turn out to be wrong. Using this idea, we start our construction of δ', the transition function for the automaton M', as follows.

(i) If $\delta(q,a)\ni(q',d)$, for some $q,q'\in Q$, $a\in\Sigma$, and $d\in\{-1,0,+1\}$, then $\delta'(q,a)\ni(q',d)$.
(ii) If $\delta(q,\vdash)\ni(q',d)$, for some $q,q'\in Q$ and $d\in\{0,+1\}$, then $\delta'(q,\triangleright)\ni(q',d)$.
(iii) If $\delta(q,\dashv)\ni(q',d)$, for some $q,q'\in Q$ and $d\in\{-1,0\}$, then $\delta'(q,\triangleleft)\ni(q',d)$.
(iv) If $\delta(q,a^{\bullet})\ni(q',d)$, for some $q,q'\in Q$, $a\in\Sigma$, and $d\in\{-1,0,+1\}$, then $\delta'(q,a^{\square})\ni(q',d)$.
(v) If $\delta(q,\vdash^{\bullet})\ni(q',d)$, for some $q,q'\in Q$ and $d\in\{0,+1\}$, then $\delta'(q,\vdash)\ni(q',d)$.
(vi) If $\delta(q,\dashv^{\bullet})\ni(q',d)$, for some $q,q'\in Q$ and $d\in\{-1,0\}$, then $\delta'(q,\dashv)\ni(q',d)$.

As soon as M moves its pebble from the p-th position to the right, M' has to travel from the p-th segment to the next, *i.e.*, the $(p{+}1)$-th segment, and find the symbol marked by the box within this segment. Assuming that the input is in the form $w'=P(w)$, for some $w\in\Sigma^*$, this only requires to find the next symbol marked by the box lying to the right of the current input position. Recall that the $(p{+}1)$-th segment has, by assumption, the same structure; the only difference is in the position of the symbol marked with the box, which corresponds exactly to the changed position of the pebble for M. Thus, after finding the marked symbol within the neighboring segment, M' can resume the simulation.

(vii) If $\delta(q, a^{\bullet}) \ni (q', +1^{\bullet})$, for some $q, q' \in Q$ and $a \in \Sigma$, we add the following instructions:
- $\delta'(q, a^{\Box}) \ni (q'_{+1}, +1)$, where q'_{+1} is a passing-through state – a new copy of q',
- $\delta'(q'_{+1}, x) \ni (q'_{+1}, +1)$, for each $x \in \Sigma \cup \{\triangleright, \triangleleft\}$,
- $\delta'(q'_{+1}, x^{\Box}) \ni (q', 0)$, for each $x^{\Box} \in \Sigma^{\Box} \cup \{\dashv\}$.

(viii) If $\delta(q, \vdash^{\bullet}) \ni (q', +1^{\bullet})$, for some $q, q' \in Q$, then
- $\delta'(q, \vdash) \ni (q'_{+1}, +1)$.
- Transitions for q'_{+1} are defined in the same way as in the item (vii).

Similarly, if M moves the pebble to the left, M' has to travel to the previous, *i.e.*, the $(p-1)$-st segment. This is resolved symmetrically with the previous case.

(ix) If $\delta(q, a^{\bullet}) \ni (q', -1^{\bullet})$, for some $q, q' \in Q$ and $a \in \Sigma$, we add the following instructions:
- $\delta'(q, a^{\Box}) \ni (q'_{-1}, -1)$, where q'_{-1} is another new passing-through copy of q',
- $\delta'(q'_{-1}, x) \ni (q'_{-1}, -1)$, for each $x \in \Sigma \cup \{\triangleright, \triangleleft\}$,
- $\delta'(q'_{-1}, x^{\Box}) \ni (q', 0)$, for each $x^{\Box} \in \Sigma^{\Box} \cup \{\vdash\}$.

(x) If $\delta(q, \dashv^{\bullet}) \ni (q', -1^{\bullet})$, for some $q, q' \in Q$, then
- $\delta'(q, \dashv) \ni (q'_{-1}, -1)$.
- Transitions for q'_{-1} are defined in the same way as in the item (ix).

From the above construction, we get $Q' = Q \cup Q_{+1} \cup Q_{-1}$, where Q is the set of the original states in M and Q_{+1}, Q_{-1} are the sets of passing-through states, *i.e.*, the sets of two new copies of states in Q, used for traversing to the neighboring segments, introduced as q'_{+1} and q'_{-1} in the items (vii) and (ix), respectively. The initial state and the final states do not change:

(xi) $q'_{\rm I} = q_{\rm I}$, $F' = F$.

This completes the definition of M'. By a not very complicated inspection of the items (i)–(xi), it is easy to see that the above transformation does preserve determinism.

Claim. *On the input w, M can get from its initial configuration, i.e., from the state $q_{\rm I}$ with both the input head and the pebble at the left endmarker, to a state $q \in Q$ with the input head at a position h and the pebble at a position p* if and only if, *on the input $P(w)$, M' can get from its initial configuration, i.e., from the state $q'_{\rm I}$ with the input head at the left endmarker, to the same state $q \in Q$ with the input head at the h-th position of the p-th segment.*

The argument for the "⇒" part is shown by induction on the number of computation steps executed by M. The "⇐" part, instead of induction on single computation steps, uses an induction on the number of times the machine M' is in a state $q \in Q$, *i.e.*, not in a passing-through state $q \in Q_{+1} \cup Q_{-1}$. (Thus, a computation path in M' is partitioned into sections $q'_{\rm I} = q_0 \to q_1 \to \cdots \to q_i \to q_{i+1} \to \ldots,$

where $q_i \to q_{i+1}$ is a section beginning and ending in states $q_i, q_{i+1} \in Q$, such that all states in between are passing-through states, from $Q_{+1} \cup Q_{-1}$. If there are no states in between, $q_i \to q_{i+1}$ represents a single step.)

As a consequence of this claim, if M accepts $w \in \Sigma^*$ by reaching an accepting state $q \in F$, then M' can reach the same state $q \in F = F'$ on the input $w' = P(w) \in \Sigma'^*$, and hence M' accepts $P(w)$. Conversely, if M' accepts $P(w)$, *i.e.*, if M' can reach an accepting state $q \in F' = F$ on the input $P(w)$, then, since q is not a passing-through state, M can reach the same state q on the input w, and hence M accepts w. □

Now we shall show a linear translation in the opposite direction.

Theorem 3.2. (a) *For each classical* 2NFA $N = (Q, \Sigma', \delta, q_{\text{I}}, F)$ *with* n *states, there exists a* PEBBLE$_1$-2NFA $N' = (Q', \Sigma, \delta', q'_{\text{I}}, F')$ *with at most* $5{\cdot}n$ *states such that, for each input* $w \in \Sigma^*$, N' *accepts* w *if and only if* N *accepts* $w' = P(w) \in \Sigma'^*$. *Here* $\Sigma' = \Sigma \cup \{\triangleright, \triangleleft\} \cup \Sigma^{\square}$ *and* P *denotes the mapping function defined by* (3.1).

(b) *Moreover, if* N *is deterministic, then so is* N'.

Proof. Note that N' does not have to be capable of simulating N on all strings $w' \in \Sigma'^*$. N' simulates N only on inputs in the form $w' = P(w)$, where $w \in \Sigma^*$ is its own input. Thus, N' can utilize the fact that the string $P(w)$ has the structure described by (3.1).

The basic idea is as follows. While N works within the same segment of $P(w)$, the simulation by N' is quite straightforward: the endmarkers $\vdash, \dashv$ surrounding w are interpreted as stoppers $\triangleright, \triangleleft$ in $P(w)$ and the presence of the pebble on a symbol $a \in \Sigma \cup \{\vdash, \dashv\}$ scanned by the input head of N' indicates that N reads $a^{\square} \in \Sigma^{\square}$ or the corresponding endmarker $\vdash, \dashv$. That is, the pebble placed at a position p reflects the fact that N' simulates, at the moment, N working within the p-th segment of the input $P(w)$.

(i) If $\delta(q, a) \ni (q', d)$, for some $q, q' \in Q$, $a \in \Sigma$, and $d \in \{-1, 0, +1\}$, then $\delta'(q, a) \ni (q', d)$.
(ii) If $\delta(q, \triangleright) \ni (q', d)$, for some $q, q' \in Q$ and $d \in \{0, +1\}$, then $\delta'(q, \vdash) \ni (q', d)$.
(iii) If $\delta(q, \triangleleft) \ni (q', d)$, for some $q, q' \in Q$ and $d \in \{-1, 0\}$, then $\delta'(q, \dashv) \ni (q', d)$.
(iv) If $\delta(q, a^{\square}) \ni (q', d)$, for some $q, q' \in Q$, $a \in \Sigma$, and $d \in \{-1, 0, +1\}$, then $\delta'(q, a^{\bullet}) \ni (q', d)$.
(v) If $\delta(q, \vdash) \ni (q', d)$, for some $q, q' \in Q$ and $d \in \{0, +1\}$, then $\delta'(q, \vdash^{\bullet}) \ni (q', d)$.
(vi) If $\delta(q, \dashv) \ni (q', d)$, for some $q, q' \in Q$ and $d \in \{-1, 0\}$, then $\delta'(q, \dashv^{\bullet}) \ni (q', d)$.

Each time N leaves the current segment, *e.g.*, if it moves its input head from the symbol $\triangleleft$ to the right (that is, in the next step, N will read the symbol $\triangleright$ belonging to the next segment), N' does not try to move its input head to the right from the right endmarker but, rather, it temporarily interrupts the simulation and enters a passing-through routine in which it traverses the entire input w from right to left and, during this traversal, it moves the pebble one position to the right. After that, with the input head at the left endmarker of w, N' is ready to resume the simulation on the next segment of $P(w)$. Note that the instructions defined in the item (vii) cover also three special subcases, namely, migration of the pebble from the left endmarker to the first input symbol, from the last input symbol to the right endmarker, or, for $w = \varepsilon$, from the left endmarker directly to the right endmarker.

(vii) If $\delta(q, \triangleleft) \ni (q', +1)$, for some $q, q' \in Q$, we add the following instructions:
- $\delta'(q, \dashv) \ni (q'_{-1}, -1)$, where q'_{-1} is a new copy of q' – a passing-through state searching for the pebble to the left,
- $\delta'(q'_{-1}, x) \ni (q'_{-1}, -1)$, for each $x \in \Sigma$,
- $\delta'(q'_{-1}, x^\bullet) \ni (q'_{-2}, +1^\bullet)$, for each $x \in \Sigma \cup \{\vdash\}$, where q'_{-2} is another new copy of q' – a passing-through state searching for the left endmarker,
- $\delta'(q'_{-2}, x^\bullet) \ni (q'_{-2}, -1)$, for each $x \in \Sigma \cup \{\dashv\}$,
- $\delta'(q'_{-2}, x) \ni (q'_{-2}, -1)$, for each $x \in \Sigma$,
- $\delta'(q'_{-2}, \vdash) \ni (q', 0)$.

Symmetrically, each time N leaves the current segment for the previous segment, *i.e.*, if it moves its head from the symbol $\triangleright$ to the left (after which it will read $\triangleleft$), N' interrupts the simulation and enters a routine traversing the entire input w from left to right and, during this traversal, it moves the pebble one position to the left.

(viii) If $\delta(q, \triangleright) \ni (q', -1)$, for some $q, q' \in Q$, we add the following instructions:
- $\delta'(q, \vdash) \ni (q'_{+1}, +1)$, where q'_{+1} is a new copy of q' – a passing-through state searching for the pebble to the right,
- $\delta'(q'_{+1}, x) \ni (q'_{+1}, +1)$, for each $x \in \Sigma$,
- $\delta'(q'_{+1}, x^\bullet) \ni (q'_{+2}, -1^\bullet)$, for each $x \in \Sigma \cup \{\dashv\}$, where q'_{+2} is another new copy of q' – a passing-through state searching for the right endmarker,
- $\delta'(q'_{+2}, x^\bullet) \ni (q'_{+2}, +1)$, for each $x \in \Sigma \cup \{\vdash\}$,
- $\delta'(q'_{+2}, x) \ni (q'_{+2}, +1)$, for each $x \in \Sigma$,
- $\delta'(q'_{+2}, \dashv) \ni (q', 0)$.

This gives $Q' = Q \cup Q_{-1} \cup Q_{-2} \cup Q_{+1} \cup Q_{+2}$, where Q is the original set of states and $Q_{-1}, Q_{-2}, Q_{+1}, Q_{+2}$ are four new copies of states in Q, introduced in the items (vii) and (viii). Finally,

(ix) $q'_{\text{I}} = q_{\text{I}}, \;\; F' = F$.

It is easy to see that the transformation described above preserves determinism.

The argument showing that N' accepts $w \in \Sigma^*$ if and only if N accepts $w' = P(w) \in \Sigma'^*$ is very similar to that of Theorem 3.1: by induction on the number of steps executed by N and by induction on the number of times N' is in a state $q \in Q$ (*i.e.*, not in a passing-through state), we can prove a claim saying that, on the input w, N' can reach a state $q \in Q$ with the input head at a position h and the pebble at a position p *if and only if*, on the input $P(w)$, N can reach the same state q with the input head at the h-th position of the p-th segment. □

Now we are ready to draw some consequences of the above translations.

Theorem 3.3. *If, for some function $f(n)$, each* 2NFA *with n states can be replaced by an equivalent* 2DFA *with at most $f(n)$ states (no pebbles), then each* PEBBLE$_1$-2NFA *with m states can be replaced by an equivalent* PEBBLE$_1$-2DFA *having no more than $5 \cdot f(3m)$ states.*

In particular, if $f(n) \le O(n^k)$, that is, if there exists a polynomial transformation from nondeterministic to deterministic classical two-way automata, then there must also exist a polynomial transformation, with the same degree of the polynomial, from nondeterministic to deterministic two-way automata equipped with a pebble, since $5 \cdot (3m)^k = (5 \cdot 3^k) \cdot m^k \le O(m^k)$.

Proof. By Theorem 3.1(a), each PEBBLE$_1$-2NFA M with m states accepting a language $L \subseteq \Sigma^*$ can be replaced by a classical 2NFA M' with at most $3 \cdot m$ states, accepting some other language $L' \subseteq \Sigma'^*$. However, for each input $w \in \Sigma^*$, M accepts w if and only if M' accepts $P(w) \in \Sigma'^*$. Here P denotes the mapping function defined by (3.1). By assumption, M' can be replaced by a classical 2DFA N, with at most $f(3m)$ states, equivalent to M'. Among others, M' accepts $P(w)$ if and only if N accepts $P(w)$. Now, by Theorem 3.2(b), we can replace N by a PEBBLE$_1$-2DFA N' with no more than $5 \cdot f(3m)$ states, such that N accepts $P(w) \in \Sigma'^*$ if and only if N' accepts $w \in \Sigma^*$. Thus, for each input $w \in \Sigma^*$, M accepts w if and only if N' accepts w, and hence these two pebble machines are equivalent. □

The situation for complementing nondeterministic two-way machines is similar.

Theorem 3.4. *If, for some function $f(n)$, each* 2NFA *with n states can be replaced by a* 2NFA *with at most $f(n)$ states recognizing the complement of the original language (no pebbles), then each* PEBBLE$_1$-2NFA *with m states can be replaced by a* PEBBLE$_1$-2NFA *with no more than $5 \cdot f(3m)$ states recognizing the complement.*

In particular, if $f(n) \le O(n^k)$, that is, if there exists a polynomial transformation for complementing nondeterministic classical two-way automata, then there must also exist a polynomial transformation, with the same degree of the polynomial, for complementing nondeterministic two-way automata equipped with a pebble.

Proof. The argument is very similar to the proof of Theorem 3.3, using Theorems 3.1(a) and 3.2(a) instead of Theorems 3.1(a) and 3.2(b).

First, by Theorem 3.1(a), convert the given PEBBLE$_1$-2NFA M into a classical 2NFA M' such that $w \in \Sigma^*$ is accepted by M if and only if $P(w) \in \Sigma'^*$ is accepted by M'. By assumption, M' can be replaced by a classical 2NFA N for the complement, with at most $f(3m)$ states. Among others, $P(w)$ is accepted by M' if and only if $P(w)$ is not accepted by N. By Theorem 3.2(a), replace N by a PEBBLE$_1$-2NFA N' such that $P(w)$ is not accepted by N if and only if w is not accepted by N'. Thus, an input $w \in \Sigma^*$ is accepted by M if and only if it is not accepted by N'. □

Corollary 3.5. *For each* PEBBLE$_1$-2DFA *with* m *states, there exists a* PEBBLE$_1$-2DFA *with at most* $60{\cdot}m$ *states recognizing the complement of the original language.*

Proof. The argument works in the same way as in the previous two theorems, this time we use Theorems 3.1(b) and 3.2(b), together with the fact that an n-state 2DFA can be complemented with no more than $4n$ states [12]. □

It was known that a PEBBLE$_1$-2DFA can be made halting on every input, and hence a machine for the complement can be obtained by exchanging accepting with rejecting states [4,5,26,27]. This would give a PEBBLE$_1$-2DFA with $O(m{\cdot}s^2)$ states, where m is the original number of states and s the size of the input alphabet. This way, a linear upper bound is obtained for languages over a fixed input alphabet, but not in the general case, where the alphabet size s can grow exponentially in m (see, *e.g.*, [25]). The construction using Corollary 3.5 does not depend on the size of the input alphabet. However, we conjecture that the upper bound in Corollary 3.5 (as well as the one presented in Cor. 4.5 below) can be improved, by a direct adaptation of a technique from [12].

4. Translation to more, but nested, pebbles

Taking into account the results presented above, a natural question arises, namely, if the same translation technique works for automata with 2 pebbles or more. The computational model used in complexity theory, where the pebbles can be moved around without restrictions, leads to nonregular languages. As an example, already with 2 pebbles we can easily recognize $L = \{a^n b^n c^n : n \geq 0\}$. It is not clear how to extend our translation to this model.

However, there is another, more restricted, model in the literature. In this more restricted model, there is a stack discipline: the pebbles are numbered $\bullet_1, \ldots, \bullet_\ell$. For each $i \in \{1, \ldots, \ell\}$, the level-$i$ pebble $\bullet_i$ can be moved one position to the left or right *only if* all lower-level pebbles $\bullet_1, \ldots, \bullet_{i-1}$ are placed on top of $\bullet_i$. During this move, $\bullet_1, \ldots, \bullet_{i-1}$ are traveling together with $\bullet_i$ along the input[1]. With this restriction, only regular languages are recognized, and there is a connection to first-order logic with transitive closure. (See *e.g.* [2,8,13].)

More formally, for $\ell \geq 0$, a *two-way nondeterministic finite automaton with* ℓ *nested pebbles* (PEBBLE$_\ell$-2NFA, for short) is $A = (Q, \Sigma, \delta, q_{\rm I}, F)$, where $Q, \Sigma, q_{\rm I}, F$ are defined as before, but now the transition function is of the form $\delta : Q \times ((\Sigma \cup \{\vdash, \dashv\}) \times 2^{\{\bullet_1,\ldots,\bullet_\ell\}}) \to 2^{Q\times(\{-1,0,+1\}\times 2^{\{\bullet_1,\ldots,\bullet_\ell\}})}$.

[1] Alternatively, these machines could be equipped with a pebble pushdown store, capable of containing $\bullet_1, \ldots, \bullet_\ell$, so that the pebbles can be put on and taken off the input tape. Implemented this way, the pebble $\bullet_i$ can be moved only if the pebbles $\bullet_1, \ldots, \bullet_{i-1}$ are not present on the input. For technical reasons, we introduce nested pebbles in a different way.

As an example, for $\ell = 4$, $\delta(q, a^{\bullet_1\bullet_2\bullet_4}) \ni (q', +1^{\bullet_1\bullet_2})$ is interpreted as follows[2]. If the machine is in the state q, scanning the symbol a on the input, with the pebbles $\bullet_1, \bullet_2$, and $\bullet_4$ (but not $\bullet_3$) placed at the current input tape position, then it changes its state to q' and moves its input head to the right, dragging also the pebbles $\bullet_1$ and $\bullet_2$.

Transitions in the form $\delta(q, a^A) \ni (q', d^B)$ are meaningless, if $B \nsubseteq A$, and hence not allowed: a pebble cannot move, if it is not scanned by the input head. As an additional restriction, to make pebbles *nested* in the stack-like fashion, B is always of the form $\{\bullet_1, \ldots, \bullet_i\}$, for some $i \in \{0, \ldots, \ell\}$.

A *deterministic* version, PEBBLE$_\ell$-2DFA, is defined in the usual way.

Now we are ready to show that the translation presented in Section 3 works also between the classical two-way automata (no pebbles) and the two-way machines with ℓ nested pebbles, for each $\ell \geq 0$. This requires to generalize encoding of the original input, introduced by (3.1), so that it is suitable for mutual simulations between machines with ℓ nested pebbles and machines without any pebbles. The cost, in the number of states, will be linear.

More precisely, the generalized function P_ℓ maps each input $w \in \Sigma^*$ of the given automaton M, equipped with ℓ nested pebbles, into a word $P_\ell(w) \in ((\Sigma \cup \{\triangleright, \triangleleft\}) \times 2^{\{\square_1, \ldots, \square_\ell\}})^*$ providing all possible positions for all pebbles $\bullet_1, \ldots, \bullet_\ell$, to be used by a classical automaton M', with no pebbles. That is, for $w = a_1 \ldots a_k$, the input tape $\vdash a_1 \ldots a_k \dashv$ with the pebbles $\bullet_1, \ldots, \bullet_\ell$ placed at input positions $p_1, \ldots, p_\ell \in \{0, \ldots, k{+}1\}$ is represented by a segment $\sigma^{(w)}_{p_1,\ldots,p_\ell}$, obtained from the string $\triangleright a_1 \ldots a_k \triangleleft$ by marking, for each $i = 1, \ldots, \ell$, the symbol at the position p_i by the box $\square_i$. The same symbol can be marked by several boxes, for example, $a_j^{\square_1\square_2\square_4}$ indicates that the pebbles $\bullet_1, \bullet_2$, and $\bullet_4$ (but not $\bullet_3$) are placed on top of the input symbol a_j. Again, the symbols $\triangleright$ and $\triangleleft$ are used as stoppers, to imitate the left and right endmarkers of the original input tape. Finally, the string $P_\ell(w)$ is obtained by enumerating all segments $\sigma^{(w)}_{p_1,\ldots,p_\ell}$, for all $(k{+}2)^\ell$ possible combinations of $p_1, \ldots, p_\ell \in \{0, \ldots, k{+}1\}$, varying lower-level pebble positions in more inner loops. (That is, p_1 runs in the innermost loop.)

$$P_\ell(w) \;=\; \sigma^{(w)}_{0,0,\ldots,0}\sigma^{(w)}_{1,0,\ldots,0}\cdots\sigma^{(w)}_{k+1,0,\ldots,0}\sigma^{(w)}_{0,1,\ldots,0}\cdots\;\cdots\sigma^{(w)}_{k+1,k+1,\ldots,k+1}. \tag{4.1}$$

However, the endmarkers $\vdash, \dashv$ are viewed as if equal to $\triangleright^{\square_1\ldots\square_\ell}$ and $\triangleleft^{\square_1\ldots\square_\ell}$, respectively, and hence the first and last segments are one symbol shorter, with no letters marked by a box.

Theorem 4.1. (a) *For each* $\ell \geq 0$ *and each* PEBBLE$_\ell$-2NFA $M = (Q, \Sigma, \delta, q_{\rm I}, F)$ *with* m *states, there exists a classical* 2NFA $M' = (Q', \Sigma', \delta', q'_{\rm I}, F')$ *with at most* $(2\ell{+}1)\cdot m$ *states such that, for each input* $w \in \Sigma^*$, M' *accepts* $w' = P_\ell(w) \in \Sigma'^*$ *if*

[2] For better readability, we write $a^{\bullet_1\bullet_2\bullet_4}$ instead of $[a, \{\bullet_1, \bullet_2, \bullet_4\}]$. In this notation, $[a, \emptyset]$ is written as a, with obvious interpretation: there is no pebble placed on the current input tape symbol a. The same holds for $+1^{\bullet_1\bullet_2}$, combining the movement of the input head with the pebbles.

and only if M accepts w. Here $\Sigma' = (\Sigma \cup \{\triangleright, \triangleleft\}) \times 2^{\{\square_1,\ldots,\square_\ell\}}$ and P_ℓ denotes the mapping function introduced by (4.1).

(b) *Moreover, if M is deterministic, then so is M'.*

Proof. Since the argument is a natural extension of the proof presented in Theorem 3.1, we shall skip many details.

The basic idea is as follows. If, during the simulation, M has its pebbles placed on the positions $p_1, \ldots, p_\ell \in \{0, \ldots, k{+}1\}$, M' works within the segment $\sigma^{(w)}_{p_1,\ldots,p_\ell}$ in $P_\ell(w)$. The simulation is straightforward, interpreting each box $\square_i$ as the corresponding pebble $\bullet_i$.

If, for some i, M moves the pebble $\bullet_i$ (but not $\bullet_{i+1}$) from a symbol a one position to the right, which is possible only if $\bullet_1, \ldots, \bullet_{i-1}$ are on top of $\bullet_i$, M' must scan the symbol a marked by $\square_1 \ldots \square_i$. This also implies that $p_1 = \ldots = p_{i-1} = p_i$. In this situation, M' has only to travel from the current input tape position to the right and find the first symbol b that is also marked by $\square_1 \ldots \square_i$. (This does not exclude the possibility that the symbols a or b are also marked by some other, higher-level, boxes.) Assuming that the input is formatted correctly, in the form $w' = P_\ell(w)$, for some $w \in \Sigma^*$, such symbol b is found in the correct segment $\sigma^{(w)}_{p,\ldots,p,p_{i+1},\ldots,p_\ell}$, at the correct position $p = p_i{+}1$ within this segment. This follows from the fact that the segments in $P_\ell(w)$ are enumerated for all possible values of $p_1, \ldots, p_\ell \in \{0, \ldots, k{+}1\}$, varying lower-level pebble positions in more inner loops. After finding such symbol b, M' can resume the simulation. Such search can be implemented by using a passing-through state $q'_{+1,i}$, where q' represents the state in which the simulation has to be resumed and $i \in \{1, \ldots, \ell\}$ the level of the simulated pebble movement. (*Cf.* items (vii) and (viii) in the proof of Thm. 3.1.)

The case in which M moves the pebble $\bullet_i$ to the left is resolved symmetrically, by the use of a passing-through state $q'_{-1,i}$. (*Cf.* items (ix) and (x) in Thm. 3.1.)

From the above construction, we obtain $Q' = Q \cup (Q_{+1} \cup Q_{-1}) \times \{1, \ldots, \ell\}$, and hence M' uses $(2\ell{+}1)\cdot m$ states. □

Theorem 4.2. (a) *For each $\ell \geq 0$ and each classical* 2NFA *$N = (Q, \Sigma', \delta, q_{\mathrm{I}}, F)$ with n states, there exists a* PEBBLE$_\ell$-2NFA *$N' = (Q', \Sigma, \delta', q'_{\mathrm{I}}, F')$ with at most $(4\ell{+}1)\cdot n$ states such that, for each input $w \in \Sigma^*$, N' accepts w if and only if N accepts $w' = P_\ell(w) \in \Sigma'^*$. Here $\Sigma' = (\Sigma \cup \{\triangleright, \triangleleft\}) \times 2^{\{\square_1,\ldots,\square_\ell\}}$ and P_ℓ denotes the mapping function introduced by* (4.1).

(b) *Moreover, if N is deterministic, then so is N'.*

Proof. This time the argument generalizes the basic idea used in the proof of Theorem 3.2. Now N' can utilize the fact that the string $P_\ell(w)$ has the structure described by (4.1).

While N works within the same segment $\sigma^{(w)}_{p_1,\ldots,p_\ell}$ in $P_\ell(w)$, N' has its pebbles placed on the positions $p_1, \ldots, p_\ell$. The simulation by N' is straightforward: the endmarkers $\vdash, \dashv$ are interpreted as the stoppers $\triangleright, \triangleleft$, and the pebbles $\bullet_1, \ldots, \bullet_\ell$ as the corresponding boxes $\square_1, \ldots, \square_\ell$.

If N moves from the stopper $\triangleleft$ to the right, to the next segment, N' temporarily interrupts the simulation and enters the following passing-through routine.

Let $\bullet_i$ be the lowest-level pebble *not placed* on the right endmarker $\dashv$. That is, $\bullet_1, \ldots, \bullet_{i-1}$ are placed on $\dashv$, and hence N' has to switch from the segment $\sigma^{(w)}_{k+1,\ldots,k+1,p_i,p_{i+1},\ldots,p_\ell}$ to $\sigma^{(w)}_{0,\ldots,0,p_i+1,p_{i+1},\ldots,p_\ell}$. (Recall that k denotes the length of the input w). To this aim, N' traverses the entire input w, starting from $\dashv$, dragging also the pebbles $\bullet_1, \ldots, \bullet_{i-1}$ to the left, and searching for $\bullet_i$. The pebble $\bullet_i$ is moved one position to the right in the middle of the input. After that, $\bullet_1, \ldots, \bullet_{i-1}$ are pulled to the left endmarker $\vdash$, where N' can resume the simulation. (*cf.* item (vii) in the proof of Thm. 3.2). This covers also the case of $i = 1$, in which only the pebble $\bullet_1$ is shifted one position to the right in the middle of the input. If $i = \ell+1$, *i.e.*, all pebbles are placed on $\dashv$, N' imitates the right endmarker of the input $P_\ell(w)$, without activation of any passing-through routine. Such routine can be implemented by using two passing-through states $q'_{-1,i}, q'_{-2,i}$, where q' represents the state in which the simulation has to be resumed and $i \in \{1, \ldots, \ell\}$ the level of the pebble movement.

The case in which N moves from the stopper $\triangleright$ to the left is resolved symmetrically, by the use of passing-through states $q'_{+1,i}, q'_{+2,i}$. (*cf.* item (viii) in Thm. 3.2).

This gives $Q' = Q \cup (Q_{-1} \cup Q_{-2} \cup Q_{+1} \cup Q_{+2}) \times \{1, \ldots, \ell\}$, and hence N' uses $(4\ell+1)\cdot n$ states. □

We are now ready to present the main results. The arguments mirror the respective proofs for Theorems 3.3, 3.4, and for Corollary 3.5.

Theorem 4.3. *If, for some function $f(n)$, each* 2NFA *with n states can be replaced by an equivalent* 2DFA *with at most $f(n)$ states (no pebbles), then, for each $\ell \geq 0$, each* PEBBLE$_\ell$-2NFA *with m states can be replaced by an equivalent* PEBBLE$_\ell$-2DFA *having no more than $(4\ell+1)\cdot f((2\ell+1)\cdot m)$ states.*

In particular, if $f(n) \leq O(n^k)$, that is, if there exists a polynomial transformation from nondeterministic to deterministic classical two-way automata, then there must also exist a polynomial transformation, with the same degree of the polynomial for each $\ell \geq 0$, from nondeterministic to deterministic two-way automata equipped with ℓ nested pebbles, since $(4\ell+1)\cdot((2\ell+1)\cdot m)^k = (4\ell+1)\cdot(2\ell+1)^k\cdot m^k \leq O(m^k)$.

Theorem 4.4. *If, for some function $f(n)$, each* 2NFA *with n states can be replaced by a* 2NFA *with at most $f(n)$ states recognizing the complement of the original language (no pebbles), then, for each $\ell \geq 0$, each* PEBBLE$_\ell$-2NFA *with m states can be replaced by a* PEBBLE$_\ell$-2NFA *with no more than $(4\ell+1)\cdot f((2\ell+1)\cdot m)$ states recognizing the complement.*

In particular, if $f(n) \leq O(n^k)$, that is, if there exists a polynomial transformation for complementing nondeterministic classical two-way automata, then there must also exist a polynomial transformation, with the same degree of the polynomial for each $\ell \geq 0$, for complementing nondeterministic two-way automata equipped with ℓ nested pebbles.

Corollary 4.5. *For each $\ell \geq 0$ and for each* PEBBLE$_\ell$-2DFA *with m states, there exists a* PEBBLE$_\ell$-2DFA *with at most $(4\ell+1)\cdot(2\ell+1)\cdot 4m$ states recognizing the complement of the original language.*

5. Conclusion

Already in 1978, it was conjectured by Sakoda and Sipser [22] that there must exist an exponential blow-up, in the number of states, for the transformation of the classical 2NFA's into 2DFA's. Nevertheless, this problem is still open. We have shown, by Theorem 4.3 above, that such blow-up could possibly be derived by proving, for some $\ell \geq 0$, an exponential gap between PEBBLE$_\ell$-2NFA's and PEBBLE$_\ell$-2DFA's. Even showing a less impressive lower bound for the PEBBLE$_\ell$-2NFA versus PEBBLE$_\ell$-2DFA trade-off, say, $\Omega(n^k)$ with some $k \geq 3$, would imply the same lower bound $\Omega(n^k)$ for the classical 2NFA versus 2DFA conversion. (To the best of authors' knowledge, the highest lower bound obtained so far is $\Omega(n^2)$ [6].) Since an automaton with several pebbles (nested in a stack-like fashion) is a different computational model, the argument might use some different witness languages.

Similarly, by Theorem 4.4, proving an exponential gap for the complementation of the PEBBLE$_\ell$-2NFA's, for any $\ell \geq 0$, would imply the same exponential gap for the complementation of the classical 2NFA's. This, in turn, would imply the exponential gap for the trade-off between 2NFA's and 2DFA's, and also between PEBBLE$_\ell$-2NFA's and PEBBLE$_\ell$-2DFA's (for this particular value of ℓ), since the complementation for the deterministic two-way machines is linear (namely, $4n$ states for 2DFA's, by [12], and at most $(4\ell+1)\cdot(2\ell+1)\cdot 4n$ states for PEBBLE$_\ell$-2DFA's, by Cor. 4.5).

Quite surprisingly, even though a polynomial trade-off between the classical 2NFA's and 2DFA's implies the same trade-off for the PEBBLE$_\ell$-2NFA's and PEBBLE$_\ell$-2DFA', for each $\ell \geq 1$, we still do not know whether a polynomial trade-off for automata with ℓ nested pebbles implies the polynomial trade-off for automata equipped with $\ell+1$ nested pebbles. (The problematic part is an analogue of Theorem 4.2, describing a simulation of ℓ nested pebbles, on an input resembling $P(w)$, by $\ell+1$ pebbles on w.)

The most natural related open problem is whether the translation results presented in Theorems 4.3 and 4.4 cannot be extended to two-way automata equipped with pebbles not restricted in a stack-like fashion. (The argument might be quite difficult, since such machines can accept nonregular languages.) Nevertheless, the answers to these questions might bring a deeper insight into the world of $O(\log n)$ space bounded computations, since the 2NFA's and 2DFA's with several unrestricted pebbles correspond to the complexity classes NSPACE$(\log n)$ and DSPACE$(\log n)$, respectively (see Sect. 3.2 in [27]).

Acknowledgements. We would like to thank anonymous referees for pointing us to the computational model of automata with pebbles nested in a stack-like fashion, which improved the quality of the results.

References

[1] J. Berman and A. Lingas, *On the complexity of regular languages in terms of finite automata*. Tech. Rep., Vol. 304, Polish Academy of Sciences (1977).

[2] M. Blum and C. Hewitt, Automata on a 2-dimensional tape, in *Proc. IEEE Symp. Switching Automata Theory* (1967), 155–160.

[3] C. Boyer, *A History of Mathematics*. John Wiley & Sons (1968).
[4] J.H. Chang, O.H. Ibarra, M.A. Palis and B. Ravikumar, On pebble automata. *Theoret. Comput. Sci.* **44** (1986) 111–121.
[5] R. Chang, J. Hartmanis and D. Ranjan, Space bounded computations: Review and new separation results. *Theoret. Comput. Sci.* **80** (1991) 289–302.
[6] M. Chrobak, Finite automata and unary languages. *Theoret. Comput. Sci.* **47** (1986) 149–158. (Corrigendum: *Theoret. Comput. Sci.* **302** (2003) 497–498).
[7] W. Ellison and F. Ellison, *Prime Numbers*. John Wiley & Sons (1985).
[8] J. Engelfriet and H.J. Hoogeboom, Tree-walking pebble automata, in *Jewels Are Forever, Contributions to Theoretical Computer Science in Honor of Arto Salomaa*, J. Karhumäki, H. Maurer, G. Păun and G. Rozenberg, Eds. Springer-Verlag (1999), 72–83.
[9] V. Geffert, Nondeterministic computations in sublogarithmic space and space constructibility. *SIAM J. Comput.* **20** (1991) 484–498.
[10] V. Geffert, Bridging across the $\log(n)$ space frontier. *Inform. Comput.* **142** (1998) 127–158.
[11] V. Geffert, C. Mereghetti and G. Pighizzini, Converting two-way nondeterministic unary automata into simpler automata. *Theoret. Comput. Sci.* **295** (2003) 189–203.
[12] V. Geffert, C. Mereghetti and G. Pighizzini, Complementing two-way finite automata. *Inform. Comput.* **205** (2007) 1173–1187.
[13] N. Globerman and D. Harel, Complexity results for two-way and multi-pebble automata and their logics. *Theoret. Comput. Sci.* **169** (1996) 161–184.
[14] J. Hartmanis, P. M. Lewis II and R. E. Stearns, Hierarchies of memory limited computations, in *IEEE Conf. Record on Switching Circuit Theory and Logical Design* (1965), 179–190.
[15] J. Hopcroft, R. Motwani and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley (2001).
[16] J. Hromkovič and G. Schnitger, Nondeterminism versus determinism for two-way nondeterministic automata: Generalizations of Sipser's separation, in *Proc. Internat. Colloq. Automata, Languages and Programming. Lect. Notes Comput. Sci.* **2719** (2003) 439–451.
[17] Ch.A. Kapoutsis, Deterministic moles cannot solve liveness. *J. Automat. Lang. Combin.* **12** (2007) 215–235.
[18] O.B. Lupanov, Über den Vergleich zweier Typen endlicher Quellen. *Probleme der Kybernetik* Akademie-Verlag, Berlin, in German, Vol. 6, 329–335 (1966).
[19] C. Mereghetti and G. Pighizzini, Optimal simulations between unary automata. *SIAM J. Comput.* **30** (2001) 1976–1992.
[20] F. Moore, On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Trans. Comput.* **C-20** (1971) 1211–1214.
[21] M. Rabin and D. Scott, Finite automata and their decision problems. *IBM J. Res. Develop.* **3** (1959) 114–125.
[22] W. Sakoda and M. Sipser, Nondeterminism and the size of two-way finite automata, in *Proc. ACM Symp. Theory Comput.* (1978), 275–286.
[23] A. Salomaa, D. Wood and S. Yu, On the state complexity of reversals of regular languages. *Theoret. Comput. Sci.* **320** (2004) 315–329.
[24] M. Shepherdson, The reduction of two-way automata to one-way automata. *IBM J. Res. Develop.* **3** (1959) 198–200.
[25] M. Sipser, Lower bounds on the size of sweeping automata, in *Proc. ACM Symp. Theory Comput.* (1979) 360–364.
[26] M. Sipser, Halting space bounded computations. *Theoret. Comput. Sci.* **10** (1980) 335–338.
[27] A. Szepietowski, *Turing Machines with Sublogarithmic Space*. *Lect. Notes Comput. Sci.* **843** (1994).

Communicated by V. Diekert.
Received September 8, 2009. Accepted December 21, 2010.