Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 339 (2018) 99–110

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

The Power of Writing, a Pebble Hierarchy and a Narrative for the Teaching of Automata Theory

Carolina Mejía[1,2]

Proyecto Curricular Matemáticas
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia

J. Andres Montoya, Christian Nolasco[3]

Departamento de Matemáticas
Universidad Nacional
Bogotá, Colombia

Abstract

In this work we study pebble automata. Those automata constitute an infinite hierarchy of discrete models of computation. The hierarchy begins at the level of finite state automata (0-pebble automata) and approaches the model of one-tape Turing machines. Thus, it can be argued that it is a complete hierarchy that covers, in a continuous way, all the models of automata that are important in the theory of computation. We investigate the use of this hierarchy as a narrative for the teaching of automata theory. We also investigate some fundamental questions concerning the power of pebble automata.

Keywords: Pebble automata, finite state automata.

We study a class of nonclassical automata that we call pebble automata. Pebble automata are two-way finite state automata (2DFAs) provided with the weakest form of writing ability: The ability of using along the computation a finite amount of pebbles, which are used to mark cells of the input tape.

Pebble automata are nonwriting extensions of two-way finite state automata which can be organized hierarchically: A pebble automaton provided with k pebbles is called a *k-pebble automaton* (k-2DFA). Notice that the pebble hierarchy is a hierarchy of writing power for nonwriting machines. The automata at level zero are

[1] The first author would like to thank COLCIENCIAS for the grant confered, and which allows her to develop her ph.D studies. The second author would like to thank Universidad Nacional de Colombia, this research was partially supported by Hermes Research System, project number 32083.

[2] Email: cmejiam@udistrital.edu.co

[3] Email: amontoyaa@unal.edu.co,cnolascos@unal.edu.co

https://doi.org/10.1016/j.entcs.2018.06.007

the standard 2DFAs, which are provided with zero pebbles, and which do not write at all. The automata at level k are the k-pebble automata, which can approach the ability of writing using their k pebbles. Which is the limit (sup) of the hierarchy?

We use the term ∞-pebble automata to denote the class of pebble automata that are provided with an unbounded amount of pebbles. The solely restriction is that the number of pebbles that can be placed on a single cell is upperbounded by a fixed constant that we call *cell capacity.* It is easy to check that ∞-pebble automata are equivalent to one-tape Turing machines. Thus, the pebble hierarchy allows us to introduce Turing machines as the left end of an evolutionary process that begins with finite state automata, and which is driven by the addition of pebbles (writing power, ink drops). It suggests that the studied hierarchy can be used as the axis of a narrative for the teaching of automata theory and other discrete models of computation. We discuss in some depth this topic: It is the first application that we propose for those nonclassical models of automata.

It is natural to ask: Is the hierarchy strict? If it were the case, the evolutionary process leading us from 2DFAs to Turing machines could be decomposed into an infinite series of discrete and natural steps. And then, it could be concluded that we are dealing with a powerful construction that could be used in the analysis and teaching of Turing machines and other discrete models of computation. On the other hand, if the hierarchy were finite, we would have something like a phase transition occurring at some k. This last possibility seems to be very unlikely.

It should be said, at this point, that the hierarchy is strict, and that it was known previous to this work. We present a new proof of this fact, and we use this proof to solve some further questions about pebble automata and their relations with other models of automata.

Organization of the work and contributions. This work is organized into three sections. In section 1 we introduce the model of pebble automata as well as some fundamental questions that are investigated along the paper. We prove in this section a weak form of universality for pebble automata, and we gather some easy facts that allow us to show that the pebble hierarchy is strict. We study, in section 2, the relations between this hierarchy of language classes and some other interesting classes of formal languages. We show that there exist deterministic context-free languages that are not included in the hierarchy, and we introduce the nondeterministic versions of the pebble classes. We can give satisfactory answers to some of the fundamental questions about the relation between deterministic and nondeterministic classes. In section 3 we include some concluding remarks, most of them related to the use of the pebble hierarchy as a narrative for the teaching of Theory.

1 Pebble automata: A hierarchy of writing power

k-pebble automata are 2DFA's provided with k indistinguishable pebbles. Those automata can use pebbles as markers: Pebbles can be placed, sensed and lifted.

Definition 1.1 Let $k \geq 1$, a k-pebble automaton is a tuple $\mathcal{M} = (Q, q_0, A, H, \Sigma, \delta)$ such that:

(i) Q is a finite set of inner states.

(ii) $q_0 \in Q$ is the initial state.

(iii) $H \subset Q$ is the set of halting states.

(iv) $A \subset H$ is the set of accepting states.

(v) Σ is the finite input alphabet.

(vi) δ is the transition function which is a partial function from the set $Q \times \Sigma \times \{0, ..., k\} \times \{0, ..., k\}$ into the set $Q \times \{0, ..., k\} \times \{0, ..., k\} \times \{\leftarrow, \triangle, \rightarrow\}$. Let $(q, a, r, s) \in Q \times \Sigma \times \{0, ..., k\} \times \{0, ..., k\}$, tuple (q, a, r, s) encodes a configuration of $\mathcal{M}$, the configuration given by:

The inner state is state q, the scanned cell is filled with the letter a, a stack of r pebbles is placed on this cell, and the automaton $\mathcal{M}$ is carrying with s pebbles in its bag.

Then, it follows that if $(q, a, r, s) \in dom(\delta)$, the inequality $r + s \leq k$ must hold. Moreover, If $\delta(q, a, r, s) = (p, n, m, d)$, we have that $\mathcal{M}$ changes its inner state from q to p, it leaves n pebbles on the scanned cell, keeps m pebbles on its bag (the equality $r + s = n + m \leq k$ must hold) and moves in the direction indicated for d (the symbol $\triangle$ forces the workhead to stay at the same cell).

Notice that pebble automata are *quasi-writing machines.* Thus, it can make a big difference if we allow the workhead to visit the empty cells of the input tape. The pebble automata that cannot visit the empty cells are called *standard,* while the pebble automata that are allowed to use all those cells are called *unrestricted.*

We want to study the power of pebble automata when they are used as language acceptors. Thus, we define the following classes of formal languages

Definition 1.2 Let L be a language, we have that $L \in \mathcal{P}_k$, if and only if, language L can be recognized using a standard k-pebble automaton..

We use the symbol $\mathcal{P}$ to denote the set $\bigcup_{i \in \mathbb{N}} \mathcal{P}_i$. Standard pebble can be easily simulated by deterministic linear bounded automata (DLBAs): The cell contents of pebble automata, which are pairs constituted by a stack of pebbles and an input letter, can be suitably encoded into the work alphabets of linear automata. It can also be checked that if $P \neq NP$, the later containment is strict: The set $\mathcal{P}$ is contained in *ptime*, while the problem SAT can be solved by deterministic linear bounded automata. Thus, if $\mathcal{DCSL}$ denotes the set constituted by all the deterministic context-sensitive languages, we have that the containment $\mathcal{P} \subseteq \mathcal{DCSL}$ holds, and if $P \neq NP$, the later containment is strict.

Let $\mathcal{M}$ be an unrestricted pebble automaton. We notice that automaton $\mathcal{M}$ can use its pebbles to simulate counters. It follows that 4-pebble unrestricted automata can simulate four counters, and as a consequence they can simulate two pushdown stacks [5]. Recall that the class of automata provided with two pushdown stacks constitutes an universal model of computation. We are interested in models of

automata which are strictly weaker than Turing machines. Therefore, we focus our attention on the class of standard pebble automata. From now on, we use the term pebble automata to refer the more restrictive notion of standard pebble automata.

What does it happens if we drop the finiteness constraint, and we consider automata provided with an unbounded amount of pebbles? We call ∞-pebble automata to the two-way automata that are provided with an infinite amount of pebbles. The solely restriction that we impose on those automata is that given an ∞-pebble automaton $\mathcal{M}$, there exists k (the *pebble capacity* of $\mathcal{M}$) such that no more than k pebbles can be stacked on the same cell. It is easy to check that the standard version of this new model of automata is equivalent to the model of deterministic linear bounded automata. It is also easy to check that the unrestricted version is equivalent to the model of one-tape Turing machines. Thus, beginning with 2DFAs, and pebble by pebble, we arrived to an universal model of computation. We think that it has an enormous teaching potential, most of this teaching potential relies on the following fact:

Beginning with a natural and easy to describe model of computation, we can deploy a hierarchy of automata that approaches (and converges to) an universal model of computation. Moreover, the transition between two successive levels of the hierarchy is given in a natural way by the addition of an epsilon of writing power (a pebble).

It is important to stress that the teaching potential of the construction cannot be reduced to the above fact. It happens that some fundamental phenomena (ideas) of computer science begin to emerge at the lowest levels. It also happens that most of the fundamental questions of theoretical computer science begin to arise when one thinks seriously about the lowest levels of the pebble hierarchy.

Let us check some easy facts related to the very first levels.

We know that $\mathcal{P}_0$ is the set of regular languages. The Blum-Hewitt theorem implies that $\mathcal{P}_1 = \mathcal{P}_0$ [1]. The language of squares, which is the language

$$\{ww : w \in \{0,1\}^*\},$$

can be easily recognized with 2 pebbles, and hence two pebbles are enough to recognize some languages that are not context-free. The set

$$\{1^p : p \text{ is a prime number}\},$$

can be recognized with three pebbles. With four pebbles one can recognize the set

$$\left\{1^i 0 1^j 0 1^{ij} : i, j \geq 0\right\},$$

and it means that four pebbles allows one to compute the general exponential function. More complex sets can be recognized with the addition of some few pebbles.

An important fact about pebble automata is that those automata are an elementary model of programmable machines (unlike 2DFA's). Notice that the ability

of searching a transition table is close to the ability of searching for a substring that matches a given pattern. Pebble automata are well suited for pattern matching tasks. When simulating a transition table a little bit of writing power could become necessary, but it happens that pebble automata can use some of their pebbles to write short strings. Thus, it seems that pebble automata can use their marking ability to search and simulate transition tables. We have the following theorem.

Theorem 1.3 *Given $n, k \geq 1$, there exists $N \in O(k + \log(n))$, and there exists a N-pebble automaton $\mathcal{M}_{n,k}$ such that any k-pebble automaton with at most n states can be simulated by $\mathcal{M}_{n,k}$.*

Proof. Let $k, n \geq 1$, it is not hard to adapt the standard construction for universal Turing machines to get, for some $N \in O(k + \log(n))$, the construction of a N-pebble automaton $\mathcal{M}_{n,k}$ that can be programmed to simulate any k-pebble automaton with at most n states.

Let $\mathcal{M}$ be a k-pebble automaton with n states, and let w be an input of $\mathcal{M}$. We can suppose that the set of states of $\mathcal{M}$ is the set $\{1, ..., n\}$. Automaton $\mathcal{M}_{n,k}$ uses k pebbles to simulate the k pebbles of $\mathcal{M}$, it uses $O(1)$ pebbles to search for the transition table of $\mathcal{M}$ which is encoded as part of the input, and it uses $O(\log(n))$ pebbles to keep track of the state of $\mathcal{M}$. The later task is accomplished in the following way:

Automaton $\mathcal{M}_{n,k}$ uses $O(\log(n))$ pebbles to write down the binary code of the actual state of $\mathcal{M}$, the workspace employed are the first $O(\log(n))$ cells of its input tape. The encoding is fairly simple: Occupied cell is equal to one, empty cell is equal to zero. □

Notice that if $\mathcal{N}$ is a n-state 2DFA, then $\mathcal{N}$ can execute at most n programs: Each program corresponds to choose an state as the initial state. On the other hand, automaton $\mathcal{M}_{n,k}$ can run an infinite number of programs: Observe that there are no restrictions, in the statement of the above theorem, related to the alphabet size of the automata that can be simulated by $\mathcal{M}_{n,k}$. This later fact clearly means that $\mathcal{M}_{n,k}$ can simulate an infinite number of pebble automata. Thus, we can conclude that the universality (self-reference) phenomenon begins to emerge at the lowest levels of our hierarchy. However, universality is not achieved at some of the levels of the hierarchy, it is achieved at the limit (like in set theoretical forcing [4]).

We want to convince the reader of the enormous teaching potential of pebble automata. Therefore, it becomes important to point out that some sophisticated problem-solving (programming) strategies can be implemented at the lowest levels of the pebble hierarchy. We will elaborate on this point in next subsection.

1.1 Programming strategies related to palindromes

The language of palindromes, which is the language

$$Pal = \left\{ w \in \{0,1\}^* : w = w^R \right\},$$

has played a very important role in the theory of formal languages [8]. Notice that the ability of recognizing palindrome is close to the abilities that are required to search a transition table. We know that Pal is a nonregular language, and that it cannot be recognized with a single pebble. It is easy to check that Pal can be recognized using two pebbles and a naive *zig-zag* strategy. Let $i \geq 1$, and let Pal^i be the language

$$\{w_1 \cdots w_i : |w_1|, ..., |w_i| \geq 2 \ \& \ w_1, ..., w_i \in Pal\}.$$

How many pebbles requires the recognition of Pal^i? A first plausible guess is that Pal^i requires $i+1$ pebbles. However, it can be checked that Pal^2 can be recognized with two pebbles.

Theorem 1.4 $Pal^2 \in \mathcal{P}_2$.

Proof. We will describe a 2-pebble automaton $\mathcal{M}$ that recognizes the language Pal^2.

Let w be the input string of $\mathcal{M}$. We suppose the following: Automaton $\mathcal{M}$ has checked for all $i \leq j$ that either $w[1, ..., i] \notin Pal$ or $w[i+1, ..., |w|] \notin Pal$. We also suppose that the pebbles are located at cells 1 and $j+1$.

The automaton begins to move the pebbles alternatively, the pebble on the left is moved rightward, and the pebble on the right is moved leftward, by the way the automaton compares the contents of the occupied cells. If a mismatch is found, then it moves the pebbles on the opposite direction until the left pebble reaches the leftend of the input tape. Notice that the right pebble has been placed on cell $j+1$, and it has been checked that $w[1, ..., j+1] \notin Pal$. Then, the right pebble is moved to cell $j+2$. Notice that the inductive condition has been preserved.

Suppose that no mismatch has been found and the two pebbles are placed on the same cell. It means that $w[1, ..., j+1] \in Pal$. Then, the automaton moves the pebbles alternatively on the opposite direction until the left pebble reaches the leftend of the input tape. The right pebble is on cell $j+1$, and it is time to check if $w[j+2, ..., |w|]$ belongs to Pal. The right pebble is moved to cell $j+2$, while the left pebble is moved to cell $|w|$. Then, a zig-zag strategy is employed to check if $w[j+2, ..., |w|] \in Pal$. If this last checking is positive the automaton halts and accepts the input, otherwise it manages to place one of the pebbles on cell $j+3$ and the other one on cell 1. Once again the inductive condition has been preserved, and hence it becomes clear that such an automaton correctly recognizes the language Pal^2. □

How many pebbles are required to recognize the language Pal^3?

We can use a running time argument to show that two pebbles are not enough. First, we observe that pebble automata are a special type of restricted one-tape Turing machine. It happens that any one-tape Turing machine recognizing Pal^3 requires time $\Omega\left(n^4\right)$. And, to finish with the argument, we have that the running time of a terminating 2-pebble automaton is $O\left(n^3\right)$.

Let $i \leq j$, notice that the number of pebbles required to recognize the language

Pal^i is at most as large as the number of pebbles required to recognize Pal^j. Thus, we have that for all $i \geq 3$, the language Pal^i cannot be recognized with two pebbles. We can prove that three pebbles are enough for the recognition of the languages Pal^3 and Pal^4. To this end, we use a *divide and conquer* strategy.

Theorem 1.5 *The languages Pal^3 and Pal^4 can be recognized with three pebbles.*

Proof. We make the proof for Pal^4.

Let w be the input string. Suppose that a single pebble is placed on the input tape, and suppose that it is placed on cell i, call this pebble *the reference pebble.* The automaton uses the remaining two pebbles to check if $w[1, ..., i]$ belongs to Pal^2, and hence, if the checking is positive, it uses the same two pebbles to check if $w[i+1, ..., |w|]$ belongs to Pal^2. If the two checkings are positive the automaton halts and accepts the input, otherwise it moves the reference pebble to cell $i+1$. The computation begins with the reference pebble placed at cell 4. The computation is rejecting if the reference pebble reaches the cell $|w| - 3$. □

The above strategy can be easily extended to prove that the language Pal^i can be recognized using $\lceil \log(i) \rceil + 1$ pebbles. We have to ask once again: How many pebbles requires the recognition of Pal^i? [4]

We are interested in proving that the number of pebbles required by Pal^i goes to infinity when i does. We can try a running time argument, it seems plausible, notice that the running time of a terminating k-pebble automaton is $O\left(n^{k+1}\right)$. However, we need an additional fact to finish with the argument, and it happens that this second ingredient fails to be true: The language Pal^i is a context-free language, and given a context-free languages L, we have that there exists an one-tape Turing machine that recognizes L in time $O\left(n^6\right)$. Thus, we have that (from the *running time point of view*) five pebbles could be enough for the recognition of all those infinitely many powers of Pal.

It is not hard to prove, although the details can be cumbersome, that the above divide and conquer strategy is an optimal one, and that for all $i \geq 2$ the language Pal^i requires $\lceil \log(i) \rceil + 1$ pebbles. The later fact implies that the pebble hierarchy is strict. It is important to stress that this later fact was proved long time ago by Hsia and Yhe [6]. They used in their proof a *nonuniform* sequence of languages that we call the HY-sequence.

In the next sections we will prove some facts concerning the relation of the pebble hierarchy with some other classes of formal languages. To this end, we use that the pebble hierarchy is infinite. However, it is important to stress that those results cannot be obtained from the proof of Hsia and Yhe [6], given the nonuniform character of their sequence (see below). All the results consigned in the next few sections can be easily obtained from the following fact:

There does not exist N such that all the powers of Pal can be recognized with N pebbles.

[4] Notice that this question is a typical lower bound question. Thus, it can be argued that some of the fundamental questions of complexity theory can be introduced in a natural way using the pebble hierarchy (pebbles as computing resources).

2 The pebble hierarchy and its relations with some other models of automata

Hsia and Yhe [6], as well as Ritchie and Springsteel [10], introduced pebble automata as computational devices that could be used to recognize context-free languages (CFLs, for short), and whose architecture is by far more elementary than the architecture of Pushdown automata (PDAs). Those two references seem to be the unique relevant references that are related to our investigations on pebble automata. Most references on pebble automata are related to two-dimensional automata and picture recognition, it includes the classical reference of Blum and Hewitt [1], where it is proved that 1-pebble automata are as powerful as 0-pebble automata. Some references refer the model of tree walking automata and tree-language recognition (see for instance [2]). Some other works study more restricted models of pebble automata that are equivalent to finite state automata (see [3] and the references herein).

2.1 Relations with context free languages

It is proved in [10] and [6] that many important classes of CFLs as the bounded, the structured and the standard CFLs can be recognized by pebble automata. Let CFL be the set constituted by all the context-free languages, the aforementioned authors left open the following question: Is CFL included in the set $\bigcup_{i\in\mathbb{N}} \mathcal{P}_i$.?

We can shed some light on this old question. First, we observe that $\bigcup_{i\in\mathbb{N}} \mathcal{P}_i$. is included in *logspace*. This later fact suggests that there exist context-free languages that cannot be recognized by pebble automata. It is widely believed that there exist (deterministic) context-free languages that do not belong to $\mathcal{L}$. Such a belief motivated the introduction of the class logDCFL (see [12]), otherwise this class would be trivially equal to $\mathcal{L}$.

Thus, it seems easy to prove that CFL is not included in $\bigcup_{i\in\mathbb{N}} \mathcal{P}_i$. If one wants to write down a detailed proof of this fact, he can try to look for a sequence of languages, say $\{L_i\}_{i\geq 2}$, satisfying the following two properties:

(i) For all $i \geq 1$, there exists j such that $L_j \in \mathcal{P}_i \backslash \mathcal{P}_{i-1}$.

(ii) The sequence is uniform. It means that there exists an alphabet Σ such that for all $i \geq 2$ it happens that $L_i \subset \Sigma^*$, and given $a \notin \Sigma$ we have that the language

$$L^{\infty} = \left\{ a^k w : w \in L_k \right\}$$

is a context free language.

If $\{L_i\}_{i\geq 2}$ is such a sequence, we have that L^{∞} is a context-free language that does not belong to $\bigcup_{i\in\mathbb{N}} \mathcal{P}_i$. The HY-sequence is nonuniform, but the sequence $\left\{ Pal^i \right\}_{i\geq 1}$ is uniform. Thus, we can prove the following

Theorem 2.1 *There exists a context-free language that is not included in $\mathcal{P}$.*

Proof. Let $a \neq 0, 1$. It remains to be proved that the language

$$Pal^{\infty} = \left\{ a^k w : w \in Pal^k \right\},$$

is a context-free language. Notice that Pal^{∞} can be easily recognized using a nondeterministic pushdown automaton similar to the one that recognizes the language Pal^*, the main difference being the following one:

Let $a^k w$ be the input string. Each time the automaton recognizes a palindromic factor of w, it makes an additional ε-transition to pop one symbol a.

If the input is $a^k w$, the automaton writes k symbols a on the pushdown stack at the beginning of the computation. The automaton accepts by empty stack. □

It seems that the language Pal^*, which is a context-free language, cannot be recognized by pebble automata. If this later conjecture were true, we could present an alternative proof of the above theorem as well as alternative proofs of corollaries 2.5 and 2.6.

2.2 Nondeterministic pebble automata

Nondeterministic pebble automata are pebble automata that can behave nondeterministically. We omit to introduce a formal definition of this new model of automata. The first question to ask: Does nondeterminism confer computational power to pebble automata?

Definition 2.2 $\mathcal{NP}_i$ is the class of languages that can be nondeterministically recognized using i pebbles.

Observe that for all $i \geq 0$

$$\mathcal{P}_i \subset \mathcal{P}_{i+1} \subseteq \mathcal{NP}_{i+1} \subseteq \mathcal{NP}_{i+2}$$

Thus, the first question concerns the nature of the containment relations between deterministic and nondeterministic classes.

The cases $i = 1, 0$ are special cases. We have that

$$\mathcal{P}_0 = \mathcal{P}_1 = \mathcal{NP}_1 = \mathcal{NP}_0$$

Next theorem allows us to answer the above question for all $i \geq 3$

Theorem 2.3 *For all $i \geq 1$, the language Pal^i can be nondeterministically recognized using three pebbles.*

Proof. Let $i \geq 3$, we define a 3-pebble automaton $\mathcal{M}_i$ that recognizes the language Pal^i. Automaton $\mathcal{M}_i$ uses nondeterminism to guess the end position of i palindromic factors. It works , on input w, as follows:

Suppose that $\mathcal{M}_i$ is located on cell j, the three pebbles are placed on cell j, the inner state of $\mathcal{M}_i$ encodes a number $k \leq i$, and $\mathcal{M}_i$ has checked that $w[1,..,j] \in Pal^k$. Suppose that $k < i$. If $j = |w|$, automaton $\mathcal{M}_i$ halts and rejects. Otherwise, it places one pebble on cell $j+1$ and advances to the right carrying the remaining two pebbles until it nondeterministically chooses a position $l > j+1$. Then, it places one pebble on cell l, and uses the other two pebbles to check if $w[j+1,...,l] \in Pal$. If the checking is positive it places the three pebbles on cell l and encodes the number $k+1$ in its inner state. □

Remark 2.4 It can also be proved, using a similar algorithm, that Pal^* belongs to $\mathcal{NP}_3$.

We can get, from the above theorem, the following corollaries:

Corollary 2.5 *For all $i \geq 3$, it happens that $\mathcal{P}_i \subsetneqq \mathcal{NP}_i$.*

Moreover, it follows that the two hierarchies are not interleaved, and nondeterminism cannot be replaced by an increasing in the number of pebbles.

Corollary 2.6 *For all $i \geq 1$, we have that $\mathcal{NP}_3 \nsubseteqq \mathcal{P}_i$.*

There are many other elementary and fundamental questions to be considered. The investigation of those questions could have an enormous teaching potential. Consider the following questions:

- Is the containment $\mathcal{P}_2 \subsetneqq \mathcal{NP}_2$ strict?
- Is the nondeterministic pebble hierarchy strict?
- How many pebbles can be simulated by nondeterminism? Or, it is better to ask: Does there exist $k > 0$ such that for all i the containment $\mathcal{P}_{i+k} \subsetneqq \mathcal{NP}_i$ holds?

3 Concluding remarks: Pebble automata and The Chomsky hierarchy

We claim that the pebble hierarchy is a powerful construction with an enormous teaching potential.

Chomsky's hierarchy has been used as the standard narrative for the teaching of automata theory. However, some criticism concerning the role of this hierarchy has been raised in recent years (for an informal discussion see [7] and [11]). It has been argued that the Chomsky hierarchy is not really a hierarchy but a small set of language classes with a linear order given by the containment relation. Moreover, it has been argued that some of the levels of this short hierarchy are unnatural, and that it is true from the machine point of view (pushdown automata), as well as from the grammar point of view (regular and general grammars).

It must be said, in favor of Chomsky's hierarchy, that it contains the language classes that occur in most applications (regular, context-free, context-sensitive and Turing-computable).

The pebble hierarchy is an infinite (somewhat continuous) hierarchy, encoding an evolutionary process that begins with the most elementary model of automata, and which converges to the universal model of one-tape Turing machines. The process is driven in a natural way by the addition of writing power. It contains (in the limit) the classes of regular, context-sensitive and Turing computable languages. But, as we have seen before, it is somewhat orthogonal to the class of context-free languages. This later fact should be considered as a major flaw of the hierarchy, a flaw that becomes even more important if one is thinking in using it as a narrative for the teaching of theory. There are two possible remedies to this problem:

(i) Some authors (see [7] and [11]) propose to include all the material concerning context-free grammars and pushdown automata into the syllabus of the courses on the theory of programming languages, and restrict the theory courses to the study of finite state automata and Turing machines. It is argued that finite state automata must be preserved as the starting point, given that this material gives all the right intuitions about nondeterminism and other fundamental ideas of computer science. We have that, from the point of view of this proposed solution, the pebble hierarchy is a complete narrative that is called to fill the gap between the starting point (finite state automata) and the arrival point (Turing machines).

(ii) We think that the material on CFGs and PDAs is so valuable that it must be (at least) approached in some way. It happens that pushdown automata can be presented as an additional branch of an evolutionary tree that begins with finite state automata, and which is driven in different ways by the addition of writing power. Recall that one can use pebbles to simulate counters and pushdown stacks. Let us elaborate a little bit on this fact.

Let us suppose that we have two set of pebbles, the first one infinite and constituted by black pebbles, while the second one is finite and constituted by white pebbles. Let us also suppose that the input tape contains a special *black* cell, say cell 0, such that any finite number of black pebbles can be placed on it. White pebbles are the pebbles that can be placed on the cells that are different to cell 0. One can use cell 0 and the unbounded provision of black pebbles to simulate a counter. Thus, to begin with, we can recover the model of two-way counter automata. If we allow to have two black cells with unbounded pebble capacity, we get the model of two-counter automata. Recall that two counters can be used to simulate a pushdown stack [5]. Thus, we can also recover the model of two-way pushdown automata. We think that one-way pushdown automata can be introduced at this point, as well as some material on context-free grammars. Moreover, it could be interesting (from the teaching point of view) to ask some questions related to the computational power of pebble automata with a larger number of black cells. We notice that four black cells could be used to simulate two pushdown stacks, and as a consequence we have that this new model of automata is an universal one [5]. It implies that a larger (and bounded) number of black cells do not confer computational power. What does it happens if all the cells are black cells?

This new model of *black automata* is at least as powerful as the model of ∞-pebble automata, and at most as powerful as the model of two-dimensional Turing machines [9]. The later two facts indicates that the model of black automata is equivalent to the Turing machine model, and we have arrived, once again, to an universal model of computation.

Acknowledgement

The first author would like to thank COLCIENCIAS for the grant confered, and which allows her to develop her Ph.D studies. The second author would like to thank Universidad Nacional de Colombia, this research was partially supported by Hermes Research System, project number 32083.

References

[1] M. Blum, C. Hewitt. Automata on a 2-Dimensional Tape. SWAT (FOCS) 1967: 155-160.

[2] M. Bojánczyk, M. Samuelides, T. Schwentick, L. Segoufin. Expressive Power of Pebble Automata. ICALP(1)2006:157-168.

[3] D Casas, J. A. Montoya. On the Real-state Processing of Regular Operations and The Sakoda-Sipser Problem. Electr. Notes Theor. Comput. Sci. 321: 23-39, 2016.

[4] L. Halbeisen. *Combinatorial Set Theory With a Gentle Introduction to Forcing.* Springer Verlag, London, 2014.

[5] J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., 2002.

[6] P. Hsia, R. Yeh. Marker automata. Inf. Sci. 8(1): 71-88, 1975.

[7] https://rjlipton.wordpress.com/2016/08/09/do-results-have-a-teach-by-date/

[8] J. A. Montoya. Open Problems Related to Palindrome Recognition: Are There Open Problems Related to Palindrome Recognition? Journal of Automata, Languages and Combinatorics 20(1): 5-25, 2016.

[9] C. Papadimitriou. *Computatational Complexity.* Addison-Wesley, Reading, Mass., 1994.

[10] R. Ritchie, F. Springsteel. Language Recognition by Marking Automata. Information and Control 20(4): 313-330, 1972.

[11] http://cstheory.stackexchange.com/questions/8539/how-practical-is-automata-theory

[12] I. Sudborough. On the Tape Complexity of Deterministic Context-Free Languages. J. ACM 25(3): 405-414, 1978.