One-Reversal Counter Machines and Multihead Automata: Revisited*

Ehsan Chiniforooshan[1], Mark Daley[1], Oscar H. Ibarra[2], Lila Kari[1], and Shinnosuke Seki[1]

[1] Department of Computer Science, University of Western Ontario, London Ontario N6A5B7, Canada
{ehsan,daley,lila,sseki}@csd.uwo.ca
[2] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. Among the many models of language acceptors that have been studied in the literature are multihead finite automata (finite automata with multiple one-way input heads) and 1-reversal counter machines (finite automata with multiple counters, where each counter can only "reverse" once, i.e., once a counter decrements, it can no longer increment). The devices can be deterministic or nondeterministic and can be augmented with a pushdown stack. We investigate the relative computational power of these machines. Our results (where C_1 and C_2 are classes of machines) are of the following types:

1. Machines in C_1 and C_2 are incomparable.
2. Machines in C_1 are strictly weaker than machines in C_2.

In obtaining results of these types, we use counting and "cut-and-paste" arguments as well as an interesting technique that shows that if a language were accepted by a device in a given class, then all recursively enumerable languages would be decidable.

1 Introduction

A deterministic pushdown automaton (DPDA) is a deterministic finite automaton (DFA) augmented with a pushdown stack. The nondeterministic versions are called NPDA and NFA, respectively. It is well-known that a DPDA is weaker than an NPDA, and the latter machines accept exactly the context-free languages. A special case of a pushdown stack is one where the stack alphabet has only two symbols, one of which is used only to mark the bottom of the stack and cannot be modified. Such a stack is called a counter. Thus, we can think of a counter as a memory unit that holds a non-negative integer (initially zero), which can be incremented by 1, decremented by 1, left unchanged and tested for zero. A DFA (NFA) augmented with multiple (at least two) counters is equivalent to a Turing machine [12]. However, if we restrict the counters

* This research was supported by the National Science Foundation Grant CCF-0524136 of Oscar H. Ibarra, and by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant R2824A01, the Canada Research Chair Award in Biocomputing to Lila Kari.

I. Černá et al. (Eds.): SOFSEM 2011, LNCS 6543, pp. 166–177, 2011.

to be *reversal-bounded*, the equivalence no longer holds. Here, reversal-bounded means that the number of alternations between non-decreasing mode and non-increasing mode each counter makes during the computation is at most r for some given positive integer r. An NFA augmented with such counters is called a *(reversal-bounded) counter machine* (NCM). The deterministic version is called a DCM. It was shown in [7] that an NCM can accept only a language whose Parikh map is semilinear. Closure and decidable properties for NCM and DCM were also investigated in [7]. For related results, see [2,3,4,5,8,10]. We note that a counter making r-reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ counters each of which makes 1 reversal. Hence, we may assume that the counters in a DCM and NCM are 1-reversal.

An NFA with multiple one-way input heads is called an NMHFA. The deterministic version is called DMHFA. These devices were first studied in [13], where it was reported that for any integer $k \geq 1$, a DMHFA (resp., NMHFA) with $k+1$ head is more powerful than one with only k heads. The proof in [13] was incomplete and was later completed in [15]. Stateless version of this hierarchy was later shown in [9].

A DPDA (resp. NPDA) can be generalized by augmenting it with multiple reversal-bounded counters – we will call it a DPCM (resp. NPCM), or by equipping it with multiple input heads – we will call it DMHPDA (resp. NMHPDA).

As for the classes of 1-reversal (pushdown) counter machines (DCM, NCM, DPCM, NPCM), our primary interest lies in the following questions:

1. whether a pushdown stack or non-determinism strictly strengthens 1-reversal counter machines, and
2. whether there is a trade-off, that is, whether an NCM can simulate a DPCM, and vice-versa.

We exhibit two languages which are in the symmetric difference of NCM and DPCM to answer the above questions negatively (Corollary 2).

It is of particular interest for applications to determine whether a given language is in DCM (or in DPCM) or not. This is because these deterministic classes possess desirable properties about decidability which are not shared by their non-deterministic counterparts such as the decidability of equivalence between a DCM and a DPCM (Corollary 5.4 in [7]) or between two DPDAs [14]. A typical tool to prove that a language is not in a given class is the pumping lemma, and actually pumping lemmas are available for DFA(= NFA), DPDA, and NPDA [6,16,17]. We propose another type of tool (Lemma 4), which reduces a language that is accepted by a DPCM with k-counters into its sub-language that is accepted by a DPCM with at most $k-1$ counters. Then, using the pumping lemma for DPDA [16], the reduction enables us to prove the existence of a language in NCM which cannot be accepted by any DPCM (Corollary 1).

In order to prove the incomparability between DPCM and NCM, we will also propose a language which is accepted by a DPCM but cannot be accepted by any NCM. A technique used toward this end deserves special mention: on the supposition that an NCM accepted this language, we could design an algorithm to decide the halting problem for Turing machines (Theorem 2).

2 Preliminaries

Let Σ be an alphabet and by Σ^* we denote the set of all words over Σ including the empty word ϵ. Let $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. For a word $w \in \Sigma^*$, $|w|$ denotes its length and w^R denotes its reverse. By $|w|_a$, we denote the number of a letter $a \in \Sigma$ occurring in w. A subset L of Σ^* is called a *language*.

Let us recall the definition of reversal-bounded (pushdown) counter machines [7]. A *reversal-bounded counter machine* is a finite automaton augmented with reversal-bounded counters. We can further augment a reversal-bounded counter machine with a pushdown stack to obtain a *reversal-bounded pushdown counter machine*. Formally, a pushdown k-counter machine M is represented by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q, Σ, Γ, F are the respective sets of states, input letters, stack symbols, and final states, q_0 is the initial state, and $Z_0 \in \Gamma$ is the particular stack symbol called the start symbol. We define the transition δ in a way that is different from but equivalent to the convention as a relation from $Q \times \Sigma \times \Gamma \times \{0,1\}^k$ into $Q \times \{\mathtt{S}, \mathtt{R}\} \times \Gamma^* \times \{-1, 0, +1\}^k$. `S` and `R` indicate the direction in which M moves its input head (`S`: stay, `R`: right). M is said to be *deterministic* if δ is a function. A configuration of M is given by a $(k+3)$-tuples $(q, w, x, c_1, \ldots, c_k)$ denoting the fact that M is in the state q, w is the "unexpended" input, the stack contains the word x, and $c_1, c_2, \ldots, c_k$ are the values contained in the k counters. Among configurations, we define a relation $\vdash_M$ as follows: $(q, aw, X\alpha, c_1, \ldots, c_k) \vdash_M (p, w', \beta\alpha, c_1 + e_1, \ldots, c_k + e_k)$ if $\delta(q, a, X, \lambda(c_1), \ldots, \lambda(c_k))$ contains $(p, d, \beta, e_1, \ldots, e_k)$, where $d \in \{\mathtt{S}, \mathtt{R}\}$, $e_1, \ldots, e_k \in \{-1, 0, +1\}$,

$$\lambda(c_i) = \begin{cases} 0 & \text{if } c_i = 0 \\ 1 & \text{otherwise,} \end{cases} \qquad \text{and} \qquad w' = \begin{cases} aw & \text{if } d = \mathtt{S} \\ w & \text{if } d = \mathtt{R}. \end{cases}$$

It is clear that the transition with the indicator `S` corresponds to the ϵ-transition in the conventional definition of pushdown counter machines, whereas that with `R` corresponds to the transition which consumes an input symbol. The reflexive and transitive closure of $\vdash_M$ is written as $\vdash_M^*$. The subscript is dropped from $\vdash_M$ and $\vdash_M^*$ whenever the particular M is understood. A word $w \in \Sigma^*$ is accepted by M if $(q_0, w, Z_0, 0, \ldots, 0) \vdash^* (q_f, \epsilon, \alpha, c_1, \ldots, c_k)$ for some $q_f \in F$. The set of all words accepted by M is denoted by $L(M)$. By ignoring the pushdown stack through the description so far, we can obtain the analogous definition and notions for counter machines (no pushdown stack).

A counter machine is said to be *reversal-bounded* if it has the property that for each of its counter the number of alternations between non-decreasing mode and non-increasing mode and vice-versa is bounded by a given constant in any computation[1]. For the reason mentioned in Introduction, we assume this

[1] The reversal-bounded property can be defined by taking into account only the accepting computation, but these two definitions are equivalent. A machine can remember, in its states, how many times each of its counters has made the reversal, and can abort a computation immediately after one of its counters makes the reversal more than the given constant times.

constant to be 1 in this paper. A finite state machine with a 1-reversal stack is said to be *1-turn*.

For an integer $k \geq 0$, let NCM(k) be the set of counter machines with k 1-reversal counters, and then let NCM $= \bigcup_{k=0}$ NCM(k). For notational conveniences, we also use NCM(k) to denote a machine in the class as well as the class of languages accepted by a machine in the class. Their deterministic subclasses are denoted by DCM(k) and DCM, respectively. Let us denote the class of pushdown machines with k 1-reversal counters (pushdown k-counter machines) by NPCM(k), and NPCM $= \bigcup_{k=0}$ NPCM(k). DPCM(k) and DPCM are their deterministic subclasses. Note that NCM(0) = DCM(0) is the class of regular languages, while NPCM(0) and DPCM(0) correspond to the classes of context-free languages and deterministic context-free languages, respectively.

3 NCM and DPCM Are Incomparable

We begin our investigation with proving that NCM and DPCM are incomparable with respect to the power to accept languages. To achieve this goal, we study the problem of how to determine whether a given language can be accepted by a machine in DCM(k), NCM(k), DPCM(k), or NPCM(k) for some $k \geq 0$ or not.

For specific values of k, pumping lemmata are available. Pumping lemmata for DCM(0) (the class of regular languages) as well as for NPCM(0) (that of context-free languages) are well-known the most [6,17]. We need the following one by Yu [16] for DPCM(0) (that of deterministic context-free languages).

Lemma 1 ([16]). *For $L \in$ DPCM(0), there exists a constant C for L such that for any pair of words $w, w' \in L$ if $w = xy$ and $w' = xz$ with $|x| > C$, and y and z begin with the same letter, then one of the following statements is true:*

- *there is a factorization $x = x_1x_2x_3x_4x_5$ with $x_2x_4 \neq \epsilon$ and $|x_2x_3x_4| \leq C$ such that for all $i \geq 0$, $x_1x_2^ix_3x_4^ix_5\{y, z\} \subseteq L$,*
- *there are factorizations $x = x_1x_2x_3$, $y = y_1y_2y_3$, and $z = z_1z_2z_3$, with $x_2 \neq \epsilon$ and $|x_2x_3| \leq C$ such that for all $i \geq 0$, $x_1x_2^ix_3\{y_1y_2^iy_3, z_1z_2^iz_3\} \subseteq L$.*

We use the above lemma along with a method which reduces a language in DPCM(k) into a language in DPCM(0) for an arbitrary k in order to prove that some language is NOT in DPCM.

Let M be a DPCM(k) for some $k \geq 0$. For an integer $1 \leq i \leq k$, we say that a word is *i-decreasing* if while M reading the word, the i-th counter is decreased.

Lemma 2. *Let $M \in$ DPCM(k). If there exists an integer $1 \leq i \leq k$ such that no word in $L(M)$ is i-decreasing, then $L(M) \in$ DPCM($k-1$).*

Proof. The basic idea is the following. It follows from the assumption that while processing an input, if M encounters the decrement of the i-th counter, the input should be rejected immediately. Also note that the transition function of M does not check the actual value of the counter, but it checks only whether the value is zero or non-zero. Thus, we can encode this zero-nonzero information about the i-th counter rather into its state, set its initial value to 0, and change it to 1 when M increments the i-th counter for the first time. □

Lemma 3. *Let $M \in \mathrm{DPCM}(k) \setminus \mathrm{DPCM}(k-1)$, and w be an i-decreasing word for some $1 \leq i \leq k$. Then $L^{(w)} := L(M) \cap w\Sigma^* \in \mathrm{DPCM}(k-1)$.*

Proof. First of all, Lemma 2 guarantees the existence of an i-decreasing word w in $L(M)$ for any $1 \leq i \leq k$. Let us build up a DPCM$(k-1)$ M' for $L^{(w)}$. Once activated, this machine firstly checks whether a given input begins with w and if not M' rejects the input immediately. During this check, M' simulates the computation of M on w except for the i-th counter. About the i-th counter, M remembers, using its state, the value of this counter which is obtained after M (deterministically) processes w. Since the i-th counter has been reversed already when M completes the processing of w, M' does not require infinite number of states in order to simulate the counter for the rest of its computation. □

By applying Lemmas 2 and 3 alternately as many times as necessary, we can find a subset of a given language in DPCM(k) which is accepted by a DPCM(0).

Lemma 4. *If $L \in$ DPCM, then there exists $w \in \Sigma^*$ such that $L^{(w)}$ is a nonempty* DPCM(0).

It is noteworthy that our argument so far does not rely on whether the pushdown stack is available for computation or not. Thus, analogous results of Lemmas 2, 3, and 4 hold for DCM.

Using Lemma 4, now we can prove the existence of a language which is in NCM, but cannot be accepted by any DPCM. As an example of such languages, we propose the *balanced language* L_b defined as follows:

$$L_b = \{a^{i_1}\#a^{i_2}\#\cdots\#a^{i_n} \mid n \geq 2, i_1, \ldots, i_n \geq 0 \\ \text{such that } i_1 + i_2 + \cdots + i_k = i_{k+1} + \cdots + i_n \text{ for some } 1 \leq k < n \}.$$

Informally speaking, an element of this language has the symbol # located at its fulcrum to the left and to the right of which there are the same number of a's. It is clear that L_b can be accepted by an NCM(1).

In order to verify that $L_b \notin$ DPCM, it suffices to prove that for any word $w \in \{a, \#\}^*$, $L_b^{(w)}$ cannot be accepted by any DPCM(0) due to Lemma 4.

Lemma 5. *For any $w \in \{a, \#\}^*$, $L_b^{(w)} = L_b \cap w\{a, \#\}^*$ is not a* DPCM(0).

Proof. Suppose $L_b^{(w)}$ were DPCM(0), then so is $L := L_b^{(w)} \cap w\Sigma^*\#\Sigma^*\#\Sigma^*$. Let $n = |w|_a$. Then $wa^C\#a^{C+n}\#, wa^C\#a^{C+n}\#a^{2(C+n)} \in L$, where C is the pumping constant given in Lemma 1. With $x = wa^C\#a^{C+n}$, $y = \#$, and $z = \#a^{2(C+n)}$, these words are written as xy and xz, which satisfy the requirements on x, y, z in Lemma 1. So one of the factorizations must hold. The pumped parts should not contain any # because any word of L contains exactly $|w|_\# + 2$ #'s. Let us consider the first factorization $x = x_1x_2x_3x_4x_5$. Since $x_2x_4 = a^m$ for some $1 \leq m \leq C$, omitting this pumpable part from $wa^C\#a^{C+n}\#a^{2(C+n)}$ would result in a word $wa^{C-i}\#a^{C+n-(m-i)}\#a^{2(C+n)}$ for some $i \geq 0$. This word is unbalanced so that other factorization $x = x_1x_2x_3$ and $y = y_1y_2y_3$ must hold.

Since $y = \#$, the pumped part y_2 is empty. If x_2 is in wa^C, then removing x_2 results in an unbalanced word. If x_2 is in a^{C+n}, then pumping x_2 more than once makes the resulting word unbalanced. □

Corollary 1. *L_b is not in* DPCM.

Proof. Suppose that $L_b \in$ DPCM. Lemma 4 implies that there exists $w \in \Sigma^*$ such that $L_b^{(w)} := L_b \cap w\Sigma^* \in$ DPCM(0), but this contradicts Lemma 5. □

Having seen that $L_b \in$ NCM $\setminus$ DPCM, now we show that there are languages in DPCM that cannot be accepted by any NCM. This result shall lead us to the incomparability of NCM and DPCM. It would also follow that the machines in DCM are strictly less powerful than those in DPCM.

We give two proofs below. The first uses the following result in [1], whose proof is quite complicated (long and tedious).

Theorem 1 ([1]). *If M is an* NCM*, then we can construct another* NCM *M' such that $L(M') = L(M)$ and M' operates in linear time (i.e., every string of length n in $L(M')$ has an accepting computation for which M' runs in linear time).*

Theorem 2. *There is a language accepted by a 1-turn* DPDA *that cannot be accepted by any* NCM.

Proof. Consider the language $L_{\text{pal}} = \{x\#x^R \mid x \in \{0,1\}^+\}$ (recall that R denotes reverse). It is obvious that L_{pal} can be accepted by a deterministic PDA (DPDA) and hence $L_{\text{pal}} \in$ DPCM(0). Suppose L_{pal} can be accepted by an NCM M with k counters. We may assume, by Theorem 1, that M operates in linear time. Consider an input $u\#u^R$, where $|u| = n$. Clearly, the number of possible configurations when the input head of M reaches $\#$ is $O(n^k)$. Now consider another input $v\#v^R$, where $|v| = n$ and $v \neq u$. It follows that since there are 2^n binary strings of length n, $u\#v^R$ will also be accepted by M for n large enough. This is a contradiction. □

We will give another proof of Theorem 2 since it employs an interesting technique, which constructs a contradictory algorithm to decide an undecidable recursively enumerable language on the assumption that a specific language can be accepted by an NCM, implying then that the language cannot be accepted by any NCM. Let us describe the details below.

1. Let $L \subseteq \{a\}^*$ be a unary recursively enumerable language that is not decidable (such L exists) and M be a Turing machine (TM) accepting L.
2. Let Q and Σ be the state set and worktape alphabet of M and let $q_0 \in Q$ be the initial state of M. Let $\Sigma' = Q \cup \Sigma \cup \{\#\}$. Note that a is in Σ. The halting computation of M on input a^d can be represented by the string $ID_1\#ID_3 \cdots \#ID_{2k-1}\#\#ID_{2k}^R \cdots \#ID_4^R\#ID_2^R$ for some $k \geq 2$ (without loss of generality, we can assume that the length of a computation is even), where $ID_1 = q_0a^d$ and ID_{2k} are the initial and halting configurations of M, and $(ID_1, ID_2, \cdots, ID_{2k})$ is a sequence of configurations of M on input a^d, i.e., configuration ID_{i+1} is a valid successor of ID_i.

Now consider the languages

$L_1 = \{ID_1\# \cdots \#ID_{2k-1}\#\#ID_{2k}^R \cdots \#ID_2^R \mid ID_{2k}$ is a halting configuration, $k \geq 2$, $ID_1 = q_0a^p (p \geq 1)$, and ID_{i+1} is a valid successor of ID_i for odd $i\}$,
$L_2 = \{ID_1\# \cdots \#ID_{2k-1}\#\#ID_{2k}^R \cdots \#ID_2^R \mid ID_{2k}$ is a halting configuration, $k \geq 2$, $ID_1 = q_0a^p (p \geq 1)$, and ID_{i+1} is a valid successor of ID_i for even $i\}$.

Clearly, L_1 and L_2 can be accepted by 1-turn DPDAs.

Theorem 3. *L_1 or L_2 cannot be accepted by any* NCM.

Proof. Suppose L_1 and L_2 are accepted by NCMs A_1 and A_2 with n_1 and n_2 1-reversal counters, respectively. We construct from A_1 and A_2 an NCM A accepting the language $L_1 \cap L_2 = \{ID_1\# \cdots \#ID_{2k-1}\#\#ID_{2k}^R \cdots \#ID_2^R \mid k \geq 2, ID_1 = q_0a^p, p \geq 1$, ID_{2k} is a halting configuration, and ID_{i+1} is a valid successor of ID_i for $i \geq 1\}$. A simulates A_1 and A_2 in parallel (hence, A will have $n = n_1 + n_2$ 1-reversal counters).

Finally, we show (using NCM A) that there exists an algorithm to decide L, which would be a contradiction. The algorithm works as follows:

1. On an input a^d, construct a finite automaton B accepting $q_0a^d\#\Sigma'^*$.
2. From the finite automaton B and the NCM A (accepting $L_1 \cap L_2$), construct an NCM A' which accepts $L(A) \cap L(B)$.
3. Test if the language accepted by A' is empty. This is possible since the emptiness problem for NCMs (or even for NPCMs) is decidable [7].

Note that $a^d \notin L$ if and only if the language accepted by A' is empty. □

From Corollary 1 and Theorem 2, we have:

Corollary 2. *1.* NCM *are strictly more powerful than* DCM.
2. NCM *and* DPCM *are incomparable.*
3. DPCM *are strictly more powerful than* DCM.

Remark 1. Note that the fact that NCM is closed under intersection and has decidable emptiness problem while this problem is undecidable for DPCM does not necessary imply that these two classes are incomparable. This kind of argument would lead to a fallacy. For example, the class DFA of finite automaton languages is closed under intersection and has decidable emptiness problem. Now consider the class Null-TM of languages defined by TMs whose inputs can only be the null string ϵ. Then the emptiness problem for Null-TM is undecidable (because the halting problem for TMs on an initially blank tape is undecidable). However, Null-TM is contained in DFA (since Null-TM consists only of the two languages $\emptyset$ and $\{\epsilon\}$), hence, not incomparable.

4 Counter Machines and Multihead Automata

A (non-deterministic) multihead finite automaton (NMHFA) M with k heads (written as NMHFA(k)) is a machine with k one-way input heads operating on an input string with a right end marker \$. At the start of the computation, the heads are on the leftmost symbol of the input string and M is in its initial state. A move of the machine is a transition of the form $\delta(q, a_1, \ldots, a_k) = \{(p, d_1, \ldots, d_k)\}$, where q is the current state, $a_1, \ldots, a_k$ are the symbols scanned by heads, p is the next state, and $d_1, \ldots, d_k$ are the movements of the heads, which can be `R` or `S` (one position to the right, or do not move). Note that the heads are *non-sensing* (i.e., the heads cannot sense the presence of the other heads on the same position). An input is accepted if M eventually enters an accepting state and all heads are on the right end marker. The machine can be augmented with a pushdown stack (NMHPDA) in the obvious way. It can be deterministic (DMHFA, DMHPDA).

The main aim of this section is to investigate the comparability and incomparability among the classes of finite state machines with those of finite state machines with multiple heads. First, we prove that a DCM can be simulated by a DMHFA. In order to prove this, we propose some properties which can be assumed for DCMs without loss of generality. The first property says that given a DCM, we can construct a DCM with the same number of counters which reads any input till its end.

Lemma 6. *Any* DCM M *can be converted to a* DCM M' *such that* $L(M') = L(M)$ *and for every input,* M' *does not go into an infinite loop on a symbol on the input tape.*

Proof. Let M have s states. M' is constructed as follows:

1. M' simulates M.
2. If M has made more than s moves while remaining on the symbol without at least one of the following happening:
   (a) a positive counter decrementing,
   (b) a zero counter becoming positive (note that when a positive counter becomes zero from being positive, it can no longer become positive again),

then M is looping on the symbol. M' then has two cases to handle:

Case 1: During the looping, M does not enter any accepting state. In this case, M' enters a special (new) reject state r and scans the remaining input in state r until the head falls off the input.

Case 2: During the looping, M enters an accepting state. In this case, M' enters a special (new) accepting state f (thus accepting the input if there is no more symbol to the right of the head). Then M', in state f, on any input enters a special rejecting state r and scans the remaining input in state r until the head falls off the input.

The next lemma provides us with one desirable property that DCMs satisfying Lemma 6 have.

Lemma 7. *Any* DCM M *can be converted into a* DCM M' *such that* $L(M') = L(M)$ *and there exist constants* c_1, c_2*, and for any input* w*, the values of counters are at most* $c_1|w| + c_2$ *during the computation by* M' *on* w*.*

Proof. Let M be a DCM(k) with s states for some $k \geq 1$ (since we concern the value of counters, we ignore the case $k = 0$). Let M' be the DCM(k) converted from M according to Lemma 6. Let $c = s \times 2^k$. Since $c \geq s$, according to the design in Lemma 6, if M' makes more than c moves while its head remaining on the same symbol, then either the event (a) or event (b) must occur.

We claim that during the computation by M' on an input of length n, the value of each counter can be at most $(c+2)^{k-1}(c+1)(n+k)$, i.e., $O(n)$, by induction on the number of counters. Note that when two counters contain the respective values n_1 and n_2, M' can increment the first counter further by cn_2 while decreasing the second one up to 0. In order to make the value of a counter as large as possible, we employ the following strategies:

1. at each transition, M' will increase the values of all counters which are in non-decreasing mode;
2. M' never decrease two counters at the same transition.

For the case $k = 1$, more than c transitions should not occur uninterruptedly without moving its head. According to the first strategy, M should increment the counter from the very beginning transition, and once being decremented, M cannot increment the counter any more. Thus the value of this counter can be at most $(c+1)n + c \leq (c+1)(n+1)$.

Now suppose that the claim holds for some $k-1$ and consider the case when M' has k counters. Let us assume that M decrements $(k-1)$-th counter for the first time while M' moving its head from the $(m-1)$-th input symbol to the m-th one. At the point, the value of $(k-1)$-th or k-th counter can reach at most $(c+2)^{k-2}(c+1)(m+k-1)$ according to the induction hypothesis, even if M' makes the other counters to be 0. While decrementing the $(k-1)$-th counter up to 0, the k-th counter can increase at most by $(c+1)\cdot(c+2)^{k-2}(c+1)(m+k-1)+c$. By expending the rest of the input $(n-m$ symbols), we can increase at most $(c+1)(n-m)+c$. Thus, the k-th counter can become largest when $m = n$ and the value is $(c+2)^{k-2}(c+1)(n+k-1)+(c+1)\cdot(c+2)^{k-2}(c+1)(n+k-1)+c$, which is at most $(c+2)^{k-1}(c+1)(n+k)$. Thus, the induction step is verified and the claim holds. By letting $c_1 = (c+2)^{k-1}(c+1)$ and $c_2 = k(c+2)^{k-1}(c+1)$, this lemma holds. □

Proposition 1. *For any* $k \geq 0$*,* DCM(k) $\subseteq$ DMHFA($2k+1$)*.*

Proof. Let $M \in$ DCM(k). Each 1-reversal counter C of M can be simulated by two input heads H_1 and H_2, which are initially positioned on the leftmost symbol of the input. When C increments by 1, H_1 is moved one position to the right. When C reverses, both heads are moved simultaneously to the right until H_1 reaches the end marker before continuing with the simulation of C. After that, decrementing C would correspond to moving H_2 one position to the right.

When H_2 reaches the end marker, this indicates that C has the value zero. This simulation works if the counter values are bounded by n, where n is the length of the input. If the counter values are bounded by $c_1 n + c_2$ for some constant c_1, c_2 (i.e., the counter values are linearly bounded), then H_1 (resp. H_2) operates modulo c_1, i.e., H_1 (resp. H_2) is moved one position to the right for every c_1 increments (resp. decrements) of C. The existence of such c_1, c_2 is guaranteed by Lemma 7. □

Next, we prove the incomparability between NPCM and NMHFA. Following the notation in the proof of Theorem 2, let us firstly consider the language:

$$L = \{ID_1 \# ID_2 \# ID_3 \# ID_4 \# \cdots \# ID_k \mid ID_1 = q_0 a^p \text{ for some } p \geq 1 \text{ and for } i \geq 1,\ ID_{i+1} \text{ is a valid successor of } ID_i\}.$$

It is clear that $L \in$ DMHFA(2). In contrast, the technique used to prove Theorem 2 is also applicable to show that $L \notin$ NPCM.

Proposition 2. *There is a language accepted by a* DMHFA(2) *that cannot be accepted by any* NPCM.

Due to Proposition 2, now it suffices to propose a language in NPCM\NMHFA to show the incomparability between NPCM and NMHFA. The marked palindromic language $L_{\text{pal}} = \{x\#x^R \mid x \in \{0,1\}^+\}$ serves this purpose, which is actually in DPDA. The following results may have already been known, but we have not been able to find an appropriate reference. We give a proof below for completeness. The proof uses a Kolmogorov-complexity-based idea in [11]. The Kolmogorov complexity of a word $w \in \Sigma^*$, denoted by $K(w)$, is defined to be the length of the shortest program that outputs only w. Let $K(w|y)$ be the conditional Kolmogorov complexity of w with respect to a given extra information y. A word w is said to be *random* if $K(w) \geq |w|$. It is well-known that there exist random strings. We state one more well-known fact without proof.

Fact 1. *If string uvw is random, then* $K(v|uw) \gg |v| - O(\log |uvw|)$.

Proposition 3. $L_{\text{pal}} \notin$ DMHFA(2).

Proof. Suppose that there were a DMHFA(2) M such that $L(M) = L_{\text{pal}}$. Let h_r, h_l be the rightmost and leftmost heads of M, respectively.

Let us consider a random word $w = w_1 w_2$ satisfying $|w_1| = |w_2| \gg \log |w| + |M|$, where $|M|$ denotes the (program) size of M. Note that $w_1 \neq w_2$ because of the randomness of w. Then we put the following input into M:

$$I_1 = *w_1 * w_2 * \# * w_2^R * w_1^R * .$$

For this input, we say that w_i $(i = 1, 2)$ is *checked* if there is a time t when h_r is on w_i^R while h_l is on w_i.

We claim that both w_1 and w_2 have to be checked. Indeed, suppose that w_1 were not checked. Consider the computation of M on I_1. Let $ID_{\text{in}}(h_r)$

($ID_{\text{out}}(h_r)$) be the configuration of M when h_r first reaches the first (resp. second) * sign in $*w_1^R*$ of I_1. The analogous notation is defined for h_l. Note that these IDs can be described of length $O(\log|w|)$. Given these IDs and w_2, we can reconstruct w_1 by a short program as follows: For a word x with $|x| = |w_1|$, let $P(x) = *0^{|w_1|} * w_2 * \# * w_2^R * x *$. We simulate M on this program $P(x)$ starting with $ID_{\text{in}}(h_r)$ and $ID_{\text{in}}(h_l)$. If $ID_{\text{in}}(h_r) \vdash^*_{P(x)} ID_{\text{out}}(h_r)$ and $ID_{\text{in}}(h_l) \vdash^*_{P(x)} ID_{\text{out}}(h_l)$ hold, then M accepts $I_1(x) = *w_1 * w_2 * \# * w_2^R * x *$. This can happen if and only if $x = w_1^R$ (otherwise, an input not in L_{pal} would be accepted by M). Therefore, $K(w_1|w_2) = O(\log|w|)$, but this contradicts Fact 1.

Now, the claim has been verified so that both w_1 and w_2 have to be checked, but clearly it cannot be done by one-way multihead FA (if w_2 is checked, then h_l is on w_2 so that it cannot return back to w_1 in order to check w_1.) □

We can easily modify the proof of Proposition 3 to prove that L_{pal} cannot be accepted by any DMHFA not depending on how many heads are available. It suffices to split the random string w into k substrings of the same length as $w = w_1 w_2 \cdots w_k$ in order to prove that any DMHFA(k) cannot accept this language. With this modification, we can verify the next proposition.

Proposition 4. $L_{\text{pal}} \notin$ DMHFA.

We further strengthen the above result by showing that even non-determinism does not help for multihead finite automata to accept L_{pal}.

Lemma 8. *Let Σ be a non-unary alphabet, and @ be a symbol that is not in Σ. For a language $L \in$ NMHFA(k), there exists a language $L' \subseteq (\Sigma \cup \{@\})^*$ in* DMHFA($k+1$) *with a property that $w \in L$ if and only if $w@\Sigma^* \cap L' \neq \emptyset$.*

Proof. The transition function of an NMHFA(k) maps a tuple $(q, a_1, a_2, \ldots, a_k)$ into a set $\{(p_1, d_{11}, d_{12}, \ldots, d_{1k}), \ldots (p_m, d_{m1}, d_{m2}, \ldots, d_{mk})\}$, where $m \geq 0$ is an integer, p_is are future states, and d_{ij}s are either `S` or `R`. Hence, if an NMHFA(k) is in configuration $(q, a_1, a_2, \ldots, a_k)$, it will have m transition choices.

If L is accepted by an NMHFA(k) M and w is a word in L, then there is a sequence $c_1, c_2, \ldots, c_{|w|}$ such that if we feed w into M and choose the c_ith transition at the ith step, we reach an accepting state. On the other hand, if $w \notin L$, there will be no such sequence. Therefore, one can build L' from L by attaching $@\langle c_1, c_2, \ldots, c_{|w|}\rangle$ at the end of every $w \in L$, where $\langle c_1, c_2, \ldots, c_{|w|}\rangle$ denotes any encoding of the sequence using the alphabet Σ. □

The above-mentioned lemma can be used to show that L_{pal} is not in NMHFA. For the sake of contradiction, suppose $L_{\text{pal}} \in$ NMHFA(k) for some k. Let L'_{pal} be the language that can be constructed from L_{pal} as in Lemma 8. Then, $L'_{\text{pal}} \in$ DMHFA($k+1$). However, an argument very similar to the one in the proof of Proposition 3 can be used to show that $L'_{\text{pal}} \notin$ DMHFA($k+1$).

Proposition 5. $L_{pal} \notin$ NMHFA.

Corollary 3. *Neither* DMHFA *nor* NMHFA *is comparable with either* DPCM *or* NPCM.

References

1. Baker, B.S., Book, R.V.: Reversal-bounded multipushdown machines. Journal of Computer and System Sciences 8, 315–332 (1974)
2. Daley, M., Ibarra, O., Kari, L.: Closure and decidability properties of some language classes with respect to ciliate bio-operations. Theoretical Computer Science 306, 19–38 (2003)
3. Duris, P., Galil, Z.: On reversal-bounded counter machines and on pushdown automata with a bound on the size of the pushdown store. Information and Control 54, 217–227 (1982)
4. Eğecioğlu, Ö., Ibarra, O.H.: On stateless multicounter machines. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 178–187. Springer, Heidelberg (2009)
5. Harju, T., Ibarra, O.H., Karhumäki, J., Salomaa, A.: Some decision problems concerning semilinearity and commutation. Journal of Computer and System Sciences 65, 278–294 (2002)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
7. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM 25, 116–133 (1978)
8. Ibarra, O.H., Eğecioğlu, Ö.: Hierarchies and characterizations of stateless multicounter machines. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 408–417. Springer, Heidelberg (2009)
9. Ibarra, O.H., Karhumäki, J., Okhotin, A.: On stateless multihead automata: Hierarchies and the emptiness problem. Theoretical Computer Science 411, 581–593 (2010)
10. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural P systems and partially blind counter machines. Natural Computing 7, 3–19 (2008)
11. Li, M., Yesha, Y.: String-matching cannot be done by a two-head one-way deterministic finite automaton. Information Processing Letters 22, 231–235 (1986)
12. Minsky, M.L.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of turing machines. Annals of Mathematics 74, 437–455 (1961)
13. Rosenberg, A.L.: On multi-head finite automata. IBM Journal of Research and Development 10, 388–394 (1966)
14. Sénizergues, G.: $l(a) = l(b)$? a simplified decidability proof. Theoretical Computer Science 281, 555–608 (2002)
15. Yao, A.C., Rivest, R.L.: $k+1$ heads are better than k. Journal of the ACM 25(2), 337–340 (1978)
16. Yu, S.: A pumping lemma for deterministic context-free languages. Information Processing Letters 31, 47–51 (1989)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer, Heidelberg (1997)