RAIRO-Theor. Inf. Appl. 58 (2024) 5
https://doi.org/10.1051/ita/2023011

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

REVERSIBLE COMPUTATIONS OF ONE-WAY COUNTER AUTOMATA

Martin Kutrib* and Andreas Malcher

Abstract. Deterministic one-way time-bounded multi-counter automata are studied with respect to their ability to perform reversible computations, which means that the automata are also backward deterministic and, thus, are able to uniquely step the computation back and forth. We study the computational capacity of such devices and obtain separation results between irreversible and reversible k-counter automata for polynomial and superpolynomial time. For polynomial time and exponential time we obtain moreover infinite and tight hierarchies with respect to the number of counters. These hierarchies are shown with Kolmogorov complexity and incompressibility arguments. In this way, on passing we can prove these hierarchies also for ordinary counter automata. This improves the known hierarchies for ordinary counter automata in the sense that we consider here a weaker acceptance condition. Then, it turns out that $k+1$ reversible counters are not better than k ordinary counters and vice versa. Finally, almost all usually studied decidability questions turn out to be undecidable and not even semidecidable for reversible multi-counter automata, if at least two counters are provided. In case of reversible one-counter automata, it is possible to show that the inclusion problem is also not even semidecidable.

Mathematics Subject Classification. 68Q45, 68Q15.

Received January 27, 2023. Accepted October 17, 2023.

1. Introduction

In the last years, reversible computational models have earned a lot of attention. The reversibility of a computation basically means that every configuration has at most one unique successor configuration and at most one unique predecessor configuration. One incentive to study such computational devices performing logically reversible computations is probably the question posed by Landauer of whether logical irreversibility is an unavoidable feature of useful computers. Landauer has demonstrated the physical and philosophical importance of this question by showing that whenever a physical computer throws away information about its previous state it must generate a corresponding amount of entropy that results in heat dissipation (see [1] for further details and references). First investigations on reversible computations have been started in the sixties of the last century both for Turing machines as well as for the massively parallel model of cellular automata. For both models it is known that irreversible computations can be made reversible. For Turing machines it is shown in the work of Lecerf [2] and Bennett [1] that for every Turing machine an equivalent reversible Turing machine

Keywords and phrases: Multi-Counter Automata, Reversible Computing, Counter Hierarchy, Kolmogorov Complexity, Computational Capacity, Decidability Questions.

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany.

* Corresponding author: kutrib@informatik.uni-giessen.de

can be constructed. For cellular automata it is known from [3] that every, possibly irreversible, one-dimensional cellular automaton can always be simulated by a reversible one-dimensional cellular automaton in a constructive way.

At the other end of the Chomsky hierarchy there are the regular languages. Here, Angluin has introduced reversible computations in deterministic finite automata (DFA) and showed that reversible DFAs are weaker than DFAs in general [4]. Moreover, it is known that two-way DFAs and reversible two-way DFAs are equally powerful [5]. Recent results on reversible regular languages concern the descriptional complexity and the minimality of reversible (one-way) DFAs and are obtained in [6–8]. Furthermore, due to their nature (real-time) quantum finite automata can be said to be inherently reversible [5, 9]. They do not capture the regular languages either. This is in contrast to modified recent definitions that lead to quantum finite automata which recognize all and only the regular languages with bounded error [10–12]. See [13] and the references therein for a recent survey on quantum automata.

For deterministic pushdown automata, the reversible variant has been introduced in [14], where it is in particular shown that the reversible variant is weaker than the general one. A special case of deterministic pushdown automata are deterministic one-counter automata where the stack alphabet consists of one symbol only, apart from the bottom symbol. Hence, the stack store can only be used to count a number of symbols and no longer to store a sequence of different symbols. In general, multi-counter automata are finite-state automata equipped with multiple counters which can be incremented, decremented, and tested for zero. It is well known that general one-way deterministic two-counter automata are computationally universal, that is, they can simulate Turing machines [15]. However, the latter simulation may need an unbounded amount of space. Hence, deterministic space-bounded, as well as time-bounded, multi-counter automata have been considered in [20] where, in particular, the case when the available time is restricted to real time is studied. The authors establish in this case an infinite and strict counter hierarchy as well as positive and negative closure results. The generalization to multi-counter automata that may work nondeterministically as well as may use two-way motion on the input tape has been done by Greibach [16]. Recent results on one-way deterministic multi-counter automata are given by Petersen in [17] where, in particular, some hierarchy results of Greibach concerning counters and polynomial time could be improved and tightened at the price of a stronger acceptance condition than defined in [16]. Finally, we already mentioned that one-counter automata can be seen as a special case of pushdown automata. Hence, multi-counter automata may be considered a special case of multi-pushdown automata introduced and studied in [18].

In this paper, we will consider reversible multi-counter automata. Such automata have been investigated by Morita in [19] with respect to universal computations. In detail, the universality result of Minsky could be improved, namely, it is shown by Morita that any Turing machine can already be simulated by a *reversible* two-counter automaton. It should be noted that, naturally, the simulation of Minsky as well as the reversible simulation of Morita may need an unbounded amount of space and time. In addition, the input has to be provided suitably encoded by using prime numbers. In this paper, we will therefore consider time-bounded (and hence space-bounded) reversible multi-counter automata that process a given plain unencoded input. The paper is organized as follows. The definition of the model and illustrating examples are given in Section 2. In Section 3, we study the computational capacity in detail and obtain as first result that there is a regular language that can clearly be accepted by irreversible k-counter automata in real time, for any $k \geq 0$, but cannot be accepted by any *reversible* k-counter automaton within time $2^{o(n)}$, regardless of the number of counters. We then prove tight counter hierarchies for reversible counter automata both for exponential time as well as polynomial time. These hierarchies are shown with Kolmogorov complexity and incompressibility arguments. In this way, on passing we can prove these hierarchies also for ordinary counter automata. This improves the known hierarchies for ordinary counter automata [17] in the sense that we consider here a weaker acceptance condition. Finally, we have incomparability results for polynomial time between reversible and irreversible counter automata if the reversible automata have strictly more counters than the irreversible once. Hence, we can draw a complete picture of the relations between the language families discussed. In Section 4, we investigate decidability questions for reversible counter automata. It turns out that all usually studied questions such as, for example, emptiness, finiteness, infiniteness, inclusion, and equivalence are undecidable and not even semidecidable for reversible

counter automata with at least two counters. In case of reversible one-counter automata it is known that emptiness, finiteness, infiniteness, and equivalence are decidable, since reversible one-counter automata can be considered as deterministic pushdown automata for which the former questions are decidable. However, we can show that the inclusion problem is also undecidable and not even semidecidable for reversible one-counter automata which improves also a result for reversible pushdown automata. The paper ends with some conclusions in Section 5.

2. Preliminaries

We denote the non-negative integers $\{0, 1, 2, \dots\}$ by $\mathbb{N}$. Let Σ^* denote the set of all words over the finite alphabet Σ. We write λ for the *empty word*, and let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. The *reversal* of a word w is denoted by w^R. For the *length* of w, we write $|w|$. The number of occurrences of a symbol $a \in \Sigma$ in $w \in \Sigma^*$ is written $|w|_a$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

Let $k \geq 0$ be an integer. A one-way k-counter automaton is a finite automaton having a single read-only input tape whose inscription is the input word between two endmarkers (we provide two endmarkers in order to have a definition consistent with two-way devices). In addition, it is equipped with k counters. At the outset of a computation the counter automaton is in the designated initial state, the counters are set to zero, and the head of the input tape scans the left endmarker. Dependent on the current state, the currently scanned input symbol, and the information whether the counters are zero or not, the counter automaton changes its state, increases or decreases the counters, and moves the input head one cell to the right or not. The automata have no extra output tape but the states are partitioned into accepting and rejecting states.

Definition 2.1. A *deterministic one-way counter automaton with $k \geq 0$ counters* (*abbreviated as* DCA(k)) is a system $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F \rangle$, where

1. Q is the finite set of *internal states*,
2. Σ is the finite set of *input symbols*,
3. $k \geq 0$ is the *number of counters*,
4. $\rhd \notin \Sigma$ is the *left* and $\lhd \notin \Sigma$ is the *right endmarker*,
5. $q_0 \in Q$ is the *initial state*,
6. $F \subseteq Q$ is the set of *accepting states*, and
7. $\delta \colon Q \times (\Sigma \cup \{\rhd, \lhd\}) \times \{+, \bot\}^k \to Q \times \{0, 1\} \times \{-1, 0, 1\}^k$ is the partial transition function that dependent on the current state, the current input symbol, and the current statuses of the counters ($+$ indicates a positive value and $\bot$ a zero). The transition function determines the successor state, the input head movement (0 means to keep the head on the current square, and 1 means to move one square to the right), and the operations on the counters (-1 means to decrease, $+1$ to increase, and 0 to keep the current value).

It is understood that the head of the input tape never moves beyond the endmarkers and that a counter value zero is never decreased.

A *configuration* of a DCA(k) $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F \rangle$ is a $(k+3)$-tuple $(q, w, h, c_1, c_2, \dots, c_k)$, where $q \in Q$ is the current state, $w \in \Sigma^*$ is the input, $h \in \{0, 1, \dots, |w| + 1\}$ is the current head position on the input tape, and $c_i \geq 0$ is the current value of counter i , $1 \leq i \leq k$. The *initial configuration* for input w is set to $(q_0, w, 0, 0, \dots, 0)$. During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash_M$.

A DCA(k) *halts* if the transition function is undefined for the current configuration (we do not require that the head has to be placed on the right endmarker in order to have a definition consistent with two-way devices). An input word w is *accepted* if the machine halts at some time in an accepting state, otherwise it is *rejected*. The *language accepted* by M is $L(M) = \{ w \in \Sigma^* \mid w \text{ is accepted by } M \}$.

Now we turn to *reversible* counter automata. Basically, reversibility is meant with respect to the possibility of stepping the computation back and forth. So, the automata have also to be backward deterministic. That is,

REV-DCA(1) forward			
(1)	$\delta(q_0, \rhd, \bot)$	$=$	$(q_1, 1, 0)$
(2)	$\delta(q_1, a, \bot)$	$=$	$(q_a, 1, 0)$
(3)	$\delta(q_1, b, \bot)$	$=$	$(q_b, 1, 0)$
(4)	$\delta(q_1, \lhd, \bot)$	$=$	$(q_f, 0, 0)$
(5)	$\delta(q_a, a, \bot)$	$=$	$(q_a, 1, +1)$
(6)	$\delta(q_a, b, \bot)$	$=$	$(q_1, 1, 0)$
(7)	$\delta(q_a, a, +)$	$=$	$(q_a, 1, +1)$
(8)	$\delta(q_a, b, +)$	$=$	$(q_a, 1, -1)$
(9)	$\delta(q_b, a, \bot)$	$=$	$(q_1, 1, 0)$
(10)	$\delta(q_b, b, \bot)$	$=$	$(q_b, 1, +1)$
(11)	$\delta(q_b, a, +)$	$=$	$(q_b, 1, -1)$
(12)	$\delta(q_b, b, +)$	$=$	$(q_b, 1, +1)$

REV-DCA(1) backward			
(1)	$\delta^{\leftarrow}(q_1, \rhd, \bot)$	$=$	$(q_0, -1, 0)$
(2)	$\delta^{\leftarrow}(q_1, a, \bot)$	$=$	$(q_b, -1, 0)$
(3)	$\delta^{\leftarrow}(q_1, b, \bot)$	$=$	$(q_a, -1, 0)$
(4)	$\delta^{\leftarrow}(q_f, \lhd, \bot)$	$=$	$(q_1, 0, 0)$
(5)	$\delta^{\leftarrow}(q_a, a, \bot)$	$=$	$(q_1, -1, 0)$
(6)	$\delta^{\leftarrow}(q_a, b, \bot)$	$=$	$(q_a, -1, +1)$
(7)	$\delta^{\leftarrow}(q_a, a, +)$	$=$	$(q_a, -1, -1)$
(8)	$\delta^{\leftarrow}(q_a, b, +)$	$=$	$(q_a, -1, +1)$
(9)	$\delta^{\leftarrow}(q_b, a, \bot)$	$=$	$(q_b, -1, +1)$
(10)	$\delta^{\leftarrow}(q_b, b, \bot)$	$=$	$(q_1, -1, 0)$
(11)	$\delta^{\leftarrow}(q_b, a, +)$	$=$	$(q_b, -1, +1)$
(12)	$\delta^{\leftarrow}(q_b, b, +)$	$=$	$(q_b, -1, -1)$

any configuration must have at most one predecessor which, in addition, is computable by a counter automaton. In particular for the read-only input tape, the machines reread the input symbol which they have been read in a preceding forward computation step. Therefore, for reverse computation steps of *one-way* machines the head of the input tape is either moved to the *left* or stays stationary. One can imagine that in a forward step, first the input symbol is read and then the input head is moved to its new position, whereas in a backward step, first the input head is moved to its new position and then the input symbol is read. In particular, for backward steps, this implies that the movement of the input head only depends on the current state. So, a DCA(k) M is said to be *reversible* (REV-DCA(k)) if and only if there exists a *reverse transition function* $\delta^{\leftarrow}\colon Q \times (\Sigma \cup \{\rhd, \lhd\}) \times \{+, \bot\}^k \to Q \times \{0, -1\} \times \{-1, 0, 1\}^k$ inducing a relation $\vdash_M^{\leftarrow}$ from a configuration to its *predecessor configuration*, so that

$$(q', w, h', c_1', c_2', \ldots, c_k') \vdash_M^{\leftarrow} (q, w, h, c_1, c_2, \ldots, c_k) \text{ if and only if}$$
$$(q, w, h, c_1, c_2, \ldots, c_k) \vdash_M (q', w, h', c_1', c_2', \ldots, c_k').$$

It is well known that general one-way two-counter automata are computational universal, that is, they can simulate Turing machines [15]. So, in the sequel we also consider restricted variants. More precisely, we consider time limits for accepting computations. Let $t\colon \mathbb{N} \to \mathbb{N}$ be a function. A DCA(k) M is said to be *t-time-bounded* or of *time complexity* t if and only if it halts on every input $w \in L(M)$ after at most $t(|w|)$ time steps. A particular time bound is *real time*, that is, the smallest time at which the counter automaton can read the input entirely (including the right endmarker). So, here real time is defined to be $t(n) = n + 2$. A DCA(k) is said to be *quasi real time* if there is a constant that bounds the number of consecutive stationary moves in all computations. Here, a stationary move is a computation step in which the input head stays on its current position.

The family of all languages which can be accepted by some device X with time complexity t is denoted by $\mathscr{L}_t(\mathsf{X})$.

To clarify our notion we continue with examples.

Example 2.2. The deterministic context-free language $\{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ is accepted by the real-time REV-DCA(1) $M = \langle Q, \{a, b\}, 1, \rhd, \lhd, \delta, q_0, \{q_f\}\rangle$ with state set $\{q_0, q_1, q_a, q_b, q_f\}$.

The basic idea of the construction is as follows. We use the counter for storing the difference between the number of a's and b's in the input. However, to enable the deterministic backward computation, the difference one is remembered in the states and the counter is only used to store larger differences. Hence, the state q_a indicates that there are more a's than b's in the input read so far and q_b denotes the opposite.

Now, the computation is started with transition (1) which moves from the left endmarker to the first symbol and enters state q_1 that indicates that the number of a's and b's currently read is equal. Then, transitions (2) and (3) are used to count the difference one. Transitions (5), (7) and (10), (12) increase the difference by one and transitions (8) and (11) decrease the difference by one. Finally, if the difference is one, then transitions (6) and (9) can be used to decrease the difference to zero, to enter state q_1, and to enter an accepting state when reading the right endmarker with transition (4). For the backward computation we just have to do the opposite by switching the roles of a and b. For example, transitions (5) and (7) of δ increase the difference by one when an a is read and there have been more a's than b's read so far. This difference is later decreased by one with transitions (6) and (8) when a b is read. Thus, for $\delta^{\leftarrow}$ we have to increase the difference when reading a b (transitions (6) and (8)) and to decrease the difference when reading an a (transitions (5) and (7)). The transitions (2)–(3) and (9)–(12) can analogously be translated. To translate the transitions (1) and (4) concerning the endmarkers is straightforward. It follows immediately from the transition function that M moves its head in any but the last computation step. So, it takes at most $n+2$ steps, that is, it works in real time. ■

Example 2.3. The non-context-free language $\{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$ can be accepted by a real-time REV-DCA(2). The basic idea is to implement the construction described in Example 2.2 twice, namely, one counter is used to check whether the number of a's is equal to the number of b's and the other counter is used to check whether the number of a's is equal to the number of c's. More precisely, the construction can be realized using the Cartesian product of the construction from Example 2.2, and one component of the state set and one counter suffices to check the difference between a's and b's (c's are ignored) and a's and c's (b's are ignored), respectively. The input is accepted if in both components the state q_1 is reached when reading the right endmarker. In this case, the numbers of a's and b's are equal as well as the numbers of a's and c's. Hence, the numbers of b's and c's are equal as well. Since the computation in each component is reversible, the overall computation is also reversible.

This idea can straightforwardly be generalized to show that the language

$$\{\, w \in \{a_1, a_2, \ldots, a_k\}^* \mid |w|_{a_1} = |w|_{a_2} = \cdots = |w|_{a_k} \,\}$$

for $k \geq 2$ and an alphabet $\{a_1, a_2, \ldots, a_k\}$ of k symbols can be accepted by a real-time REV-DCA$(k-1)$. ■

3. Computational capacity of reversible counter automata

Here, we consider the computational capacities of reversible counter automata and compare it with the general variants. First, we are interested in the role played by stationary moves in quasi real-time computations. In order to settle this role, we first deal with a more general issue. For, not necessarily reversible, counter automata the restriction to be able to add or subtract only 1 per step to or from the counters is not a limitation of the computational capacity. Clearly, any counter automaton that may add or subtract an arbitrary number to or from the counters in a single step can be simulated by a sequence of stationary moves that increment or decrement the counters by only 1 per step. However, the simulation can be done without loss of time. It has been mentioned in [20] without details. Here, we will show that the construction can be done such that reversibility is preserved.

Lemma 3.1. *Let $k, c \geq 0$ be integers. For every* REV-DCA(k) *that obeys some time complexity $t(n)$ and that has the ability to alter the value of each counter independently by any integer between $-c$ and c in a single step, an equivalent ordinary* REV-DCA(k) *obeying the time complexity $t(n)$ can effectively be constructed.*

Proof. Let $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F\rangle$ be a REV-DCA(k) that has the ability to alter the value of each counter independently by any integer between $-c$ and c in a single step. The basic idea of the construction of an equivalent ordinary REV-DCA(k) $M' = \langle Q', \Sigma, k, \rhd, \lhd, \delta', q_0', F'\rangle$ is as follows. A counter value x of M is represented by the counter value $\lfloor \frac{x}{c} \rfloor$ and a state component that stores x mod c. To this end, we set $Q' = Q \times \{0, 1, \ldots, c-1\}^k$, $q_0' =$

$(q_0,(0,0,\ldots,0))$, and $F'=F\times\{0,1,\ldots,c-1\}^k$. The transition function δ' has to be constructed, in particular, such that it is reversible. For $q\in Q$, $m_1,m_2,\ldots,m_k\in\{0,1,\ldots,c-1\}$, $a\in(\Sigma\cup\{\rhd,\lhd\})$, $d_1,d_2,\ldots,d_k\in\{+,\bot\}$, we define

$$\delta'((q,(m_1,m_2,\ldots,m_k)),a,d_1,d_2,\ldots,d_k)=((q',(m_1',m_2',\ldots,m_k')),s,b_1,b_2,\ldots,b_k) \tag{3.1}$$

if and only if

$$\delta(q,a,\hat{d}_1,\hat{d}_2,\ldots,\hat{d}_k)=(q',s,\hat{b}_1,\hat{b}_2,\ldots,\hat{b}_k) \tag{3.2}$$

and $\hat{d}_i=\bot$ if $m_i=0$ and $d_i=\bot$, and $\hat{d}_i=+$ otherwise, and

$$(m_i',b_i)=\begin{cases}(m_i+\hat{b}_i,0) & \text{if } 0\leq m_i+\hat{b}_i\leq c-1\\ (m_i+\hat{b}_i+c,-1) & \text{if } m_i+\hat{b}_i<0\\ (m_i+\hat{b}_i-c,1) & \text{if } m_i+\hat{b}_i>c-1\end{cases},$$

for $1\leq i\leq k$. Note that $-1\leq b_i\leq 1$ and $-c\leq\hat{b}_i\leq c$. Immediately, from the construction it follows that $(q_0,w,0,0,\ldots,0)\vdash_M^*(q,w,h,c_1,c_2,\ldots,c_k)$ if and only if

$$((q_0,(0,0,\ldots,0)),w,0,0,\ldots,0)\vdash_{M'}^*((q,(c_1 \bmod c,\ldots,c_k \bmod c)),w,h,\lfloor\frac{c_1}{c}\rfloor,\ldots,\lfloor\frac{c_k}{c}\rfloor).$$

So, we conclude that $L(M)=L(M')$ and that M and M' share the same time complexity. It remains to be shown that M' is reversible.

Since M is reversible, the transition from equation (3.2) can be reversed, say by

$$\delta^{\leftarrow}(q',a,\tilde{d}_1,\tilde{d}_2,\ldots,\tilde{d}_k)=(q,s,-\hat{b}_1,-\hat{b}_2,\ldots,-\hat{b}_k).$$

Then, we construct

$$\delta'^{\leftarrow}((q',(m_1',m_2',\ldots,m_k')),a,d_1',d_2',\ldots,d_k')=((q,(m_1'',m_2'',\ldots,m_k'')),s,b_1',b_2',\ldots,b_k') \tag{3.3}$$

in case of $\tilde{d}_i=\bot$ if $m_i'=0$ and $d_i'=\bot$, and $\tilde{d}_i=+$ otherwise, by setting

$$(m_i'',b_i')=\begin{cases}(m_i'-\hat{b}_i,0) & \text{if } 0\leq m_i'-\hat{b}_i\leq c-1\\ (m_i'-\hat{b}_i+c,-1) & \text{if } m_i'-\hat{b}_i<0\\ (m_i'-\hat{b}_i-c,1) & \text{if } m_i'-\hat{b}_i>c-1\end{cases},$$

for $1\leq i\leq k$.

In order to show that equation (3.3) reverses equation (3.1) we distinguish the three cases of the construction of (m_i',b_i).

Case $(m_i',b_i)=(m_i+\hat{b}_i,0)$ Here we know $b_i=0$ and $0\leq m_i+\hat{b}_i\leq c-1$. Since $m_i'-\hat{b}_i=m_i+\hat{b}_i-\hat{b}_i=m_i\in\{0,1,\ldots,c-1\}$ we derive that (m_i'',b_i') has been set to $(m_i'-\hat{b}_i,0)$. Therefore, we conclude $m_i''=m_i'-\hat{b}_i=m_i$ and $b_i'=0$. So, equation (3.3) reverses equation (3.1) in this case.

Case $(m_i',b_i)=(m_i+\hat{b}_i+c,-1)$ The condition for this case is $m_i+\hat{b}_i<0$ and we know $b_i=-1$. Since $m_i'-\hat{b}_i=m_i+\hat{b}_i+c-\hat{b}_i=m_i+c>c-1$ we derive that (m_i'',b_i') has been set to $(m_i'-\hat{b}_i-c,1)$. Therefore, we conclude $m_i''=m_i'-\hat{b}_i-c=m_i$ and $b_i'=1$. So, equation (3.3) reverses equation (3.1) also in this case.

Case $(m_i', b_i) = (m_i + \hat{b}_i - c, 1)$ Here we have $b_i = 1$ and $m_i + \hat{b}_i > c - 1$. Since $m_i' - \hat{b}_i = m_i + \hat{b}_i - c - \hat{b}_i = m_i - c < 0$ we derive that (m_i'', b_i') has been set to $(m_i' - \hat{b}_i + c, -1)$. Therefore, we conclude $m_i'' = m_i' - \hat{b}_i + c = m_i$ and $b_i' = -1$. So, equation (3.3) reverses equation (3.1) in this case, as well.

From the three cases we derive that M' is reversible. □

The next step is to use Lemma 3.1 to show that quasi real-time computations can be sped-up to real time.

Theorem 3.2. *Let $k \geq 0$ be an integer. For every quasi real-time* REV-DCA(k) *an equivalent real-time* REV-DCA(k) *can effectively be constructed.*

Proof. Let M be a quasi real-time REV-DCA(k) that never performs more than $\ell \geq 0$ stationary moves consecutively. Clearly, if $\ell = 0$ then M does not perform a stationary move at all and, thus, works in real time. So, we consider $\ell \geq 1$ in the rest of the proof.

First, we assume that M may alter the value of each counter independently by any integer between $-(\ell+1)$ and $\ell+1$ in a single step. This assumption is certainly true since, in fact, M will change its counter values by at most one in each move. However, now we can safely apply the construction of the proof of Lemma 3.1 to M and obtain an equivalent REV-DCA(k) $M' = \langle Q', \Sigma, k, \rhd, \lhd, \delta', q_0', F' \rangle$ that is able to alter the value of each counter independently by any integer between $-(\ell+1)$ and $\ell+1$ in a single step. So far, M' will not use these extended abilities, since it will change its counter values by at most one in each move, but it has the abilities. Moreover, M' works still in quasi real time.

The next step is to speed-up M' to real time. To this end, we modify M' to an equivalent real-time REV-DCA(k) $M'' = \langle Q', \Sigma, k, \rhd, \lhd, \delta'', q_0', F' \rangle$. Note that due to its construction, M' knows in each step whether the represented value of its counter c_i, for $1 \leq i \leq k$, is at least $\ell+1$. If it is strictly less than $\ell+1$ then M' knows its exact value. The purpose of the extended abilities of M' is to simulate at once a possibly empty sequence of stationary moves and a possibly subsequent non-stationary step by M''. So, M'' will be able to detect whether its counters can get empty within the next $\ell+1$ steps.

By the construction in the proof of Lemma 3.1, a counter value of M'' is represented by the sum of the counter value itself times $\ell+1$ and a state component that stores a number from $\{0, 1, \ldots, \ell\}$. So, we have $Q' = Q \times \{0, 1, \ldots, \ell\}^k$, $q_0' = (q_0, (0, 0, \ldots, 0))$, and $F' = F \times \{0, 1, \ldots, \ell\}^k$. Next, the transition function δ'' has to be constructed, in particular, such that the reversibility of M' is preserved.

Given $q \in Q'$, $a \in (\Sigma \cup \{\rhd, \lhd\})$, and $d_1, d_2, \ldots, d_k \in \{+, \bot\}$, the transition $\delta''(q, a, d_1, d_2, \ldots, d_k)$ is defined by the computation $\gamma_1 \vdash_{M'} \gamma_2 \vdash_{M'} \cdots \vdash_{M'} \gamma_m$ of M' starting on

$$\gamma_1 = (q, x, h, c_1, c_2, \ldots, c_k),$$

where $(x, h) = (\lambda, 0)$ if $a = \rhd$, and $(x, h) = (a, 1)$ otherwise, as well as $c_i = 1$ if $d_i = +$, and $c_i = 0$ if $d_i = \bot$, $1 \leq i \leq k$. The computation starts with a possibly empty sequence of stationary moves on input symbol x, followed by a non-stationary move on x. Since M' works in quasi real time this takes at most $\ell+1$ steps.

First, we construct δ'' for the cases where the computation does not halt before step $\ell+1$. Let $\gamma_m = (q', \lambda, h+1, c_1', c_2', \ldots, c_k')$ be the configuration reached after the non-stationary move. Recall that M' changes its represented counter values by at most one in each move, thus, in the computation the value of each counter c_i by some $-(\ell+1) \leq j_i \leq \ell+1$, and that the represented counter values of M' and M'' are 'compressed'. So, we conclude $-1 \leq c_i' - c_i \leq 1$. Then, the transition $\delta''(q, a, d_1, d_2, \ldots, d_k)$ of M'' to be defined yields $(q', 1, c_1' - c_1, c_2' - c_2, \ldots, c_k' - c_k)$.

Second, assume that the computation halts in configuration γ_m before step $\ell+1$, that is, before performing the non-stationary move. Then the transition $\delta''(q, a, d_1, d_2, \ldots, d_k)$ of M'' to be defined yields $(q', 0, c_1' - c_1, c_2' - c_2, \ldots, c_k' - c_k)$.

So, from the construction we obtain that, given an input w, the computation of M' is unambiguously split into sequences of steps each of which is performed by M'' at once. If M' accepts, so does M'' also in cases where

the input is accepted after some stationary moves at the end of the computation. Conversely, every step of M'' corresponds to a sequence of steps of M'. So, we have $L(M') = L(M'')$. Moreover, M'' works in real time.

Finally, the reversibility of M'' follows by the reversibility of M'. Since M' is reversible, the computation $h_n \vdash^{\leftarrow}_{M'} h_{n-1} \vdash^{\leftarrow}_{M'} \cdots \vdash^{\leftarrow}_{M'} h_1$ is unique. There remains only one point. In the backwards computation, first the non-stationary move is simulated followed by some stationary moves. While this causes no trouble in general, we have to argue that the computation does not go before the initial configuration with stationary moves. However, since a loop with stationary moves on the left endmarker that runs from the initial configuration to the initial configuration would imply that the accepted language is empty, we safely may assume that it does not exist. So, any stationary transition on the left endmarker that leads to the initial configuration can safely be removed from M''. Note, that these transitions can be identified by the construction of M''. We conclude that M'' is reversible. □

So, the family of languages accepted by quasi real-time REV-DCA(k) equals the family of languages accepted by real-time REV-DCA(k).

Next, we turn to the question of whether the property of being reversible causes weaker computing capabilities for counter automata at all. It is known that reversible two-counter automata that do not have an input tape but receive their inputs suitably encoded into their counters can simulate Turing machines [19]. So, we have to consider counter automata working within time bounds. Though reversible counter automata are able to accept even non-context-free languages in real time (Ex. 2.3), their reversibility has a drastic impact on their computational capacities for certain languages. We will show that there is a regular language not accepted by any reversible counter automaton (with an arbitrary number of counters) with the super-polynomial time complexity $2^{o(n)}$. To this end, we will use Kolmogorov complexity and incompressibility arguments. General information on this technique can be found, for example, in the textbook [21], Chapter 7. Let $w \in \{0,1\}^*$ be an arbitrary binary string. The Kolmogorov complexity $C(w)$ of w is defined to be the minimal size of a binary program (Turing machine) describing w. The following key component for using the incompressibility method is well known: there are binary strings w of *any* length such that $|w| \leq C(w)$.

Theorem 3.3. *Let $k \geq 0$ be an integer. There exists a regular language that is not accepted by any $2^{o(n)}$-time* REV-DCA(k).

Proof. We consider the regular language $L = ((aa + a)(bb + b))^*(aa + a + \lambda)$ as witness. Assume in contrast to the assertion that L is accepted by some REV-DCA(k) $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F\rangle$ in some time $t(n) = 2^{o(n)}$.

We choose a word $w \in \{0,1\}^+$ long enough such that $C(w) \geq |w|$. Now, w is encoded as follows. From left to right the digits are represented alternating by a's and b's such that a 0 is represented by a single letter and a 1 by a double letter. For example, the word 010110 is encoded as *abbabbaab*. Let $\varphi(w)$ denote the code of w. We have $\varphi(w) \in L$. Next, we consider the accepting computation on $\varphi(w)$ and show that w can be compressed.

Since M accepts in time $2^{o(n)}$, the maximum number stored in some counter of M in the accepting computation on $\varphi(w)$ is bounded from above by $2^{o(n)}$. Therefore, omitting the second component, each configuration $(q, \varphi(w), h, c_1, c_2, \ldots, c_k)$ of M can be encoded with

$$\lceil \log(|Q|)\rceil + \lceil \log(|\varphi(w)| + 2)\rceil + k \cdot o(|\varphi(w)|) = o(|\varphi(w)|) = o(|w|)$$

bits.

Knowing M, the length of $\varphi(w)$, and the accepting configuration on $\varphi(w)$ without the second component, w can be reconstructed as follows. For each candidate string x of length $|\varphi(w)|$, the REV-DCA(k) M is simulated.

We claim that if the simulation accepts in the accepting configuration of $\varphi(w)$ then we have $x = \varphi(w)$ and, thus, decoding $\varphi(w)$ yields w.

In order to show the claim, assume that $x \neq \varphi(w)$. Then the computation is run backwards as long as the suffixes of $\varphi(w)$ and x are identical, thus, reaching some configurations $(q, uzv, |u| + 2, c_1, c_2, \ldots, c_k)$ and $(q, u'z'v, |u| + 2, c_1, c_2, \ldots, c_k)$ with $uzv = \varphi(w)$ and $u'z'v = x$, $z, z' \in \{a, b\}$, $|u| = |u'|$, and $z \neq$

z'. We may safely assume that $z = a$ and $z' = b$. Since $uzbb = uabb$ belongs to L the computation continuing in $(q, uzbb, |u| + 2, c_1, c_2, \ldots, c_k)$ ends accepting. But then the computation continuing in $(q, u'z'bb, |u| + 2, c_1, c_2, \ldots, c_k)$ is accepting as well. However the input $u'z'bb = u'bbb$ has to be rejected since it ends with three b's. This contradiction shows the claim.

We conclude that the Kolmogorov complexity of w is

$$C(w) = o(|w|) + \lceil \log(|\varphi(w)|) \rceil + \ell = o(|w|),$$

for a positive constant ℓ which gives the size of M and the program that reconstructs w. So, we have $C(w) < |w|$, for w long enough. This is a contradiction since w has been chosen such that $C(w) \geq |w|$. The contradiction shows that L is not accepted by M. □

Since even DCA(0), which are essentially DFAs, can accept all regular languages, we have separated the computational capacity of DCA(k) and REV-DCA(k) for all $k \geq 1$ if they obey the time complexity $2^{o(n)}$.

Theorem 3.4. *Let $k \geq 0$ be an integer. The family of languages accepted by* REV-DCA(k) *in at most $2^{o(n)}$-time is strictly included in the family of languages accepted by* DCA(k) *in at most $2^{o(n)}$ time.*

Next, we turn to the impact of the number of counters to the computational capacities of reversible counter automata. Infinite and strict counter hierarchies for general counter automata working in real time are known for a long time [20, 22]. Generalizations to polynomial and exponential time complexities have been obtained in [17]. However, the hierarchy results [17] rely on a stronger acceptance condition. This stronger condition requires that the computations on *all* inputs have to respect the time complexity. In particular, this stronger condition weakens the non-acceptance results. On passing, here we obtain the known hierarchies also for ordinary counter automata even for the weaker acceptance condition as used overall in this paper. In our definition only accepting computations have to obey the time complexity. To obtain the results, we use once more Kolmogorov arguments.

Our next counter hierarchy concerns reversible and general counter automata working in some exponential time. In order to define languages that serve as witnesses, let $\varphi\colon \{a, b, \bar{a}, \bar{b}\}^* \to \{0, 1\}^*$ be the homomorphism defined through $\varphi(a) = \varphi(\bar{a}) = 0$ and $\varphi(b) = \varphi(\bar{b}) = 1$. Next, we consider all words w over the alphabet $\{a, b, \bar{a}, \bar{b}\}$ as binary numbers $\varphi(w)$. The integer represented by $\varphi(w)$ is denoted by $\eta(\varphi(w))$. Let $k \geq 2$ and $j \geq 1$ be integers and $\varphi(w) = z_1 z_2 \cdots z_{j \cdot k} \in \{0, 1\}^{j \cdot k}$. Then, for all $1 \leq i \leq k$, we consider the scattered factors $v_i^{(k)}(\varphi(w)) = z_i z_{k+i} z_{2k+i} \cdots z_{(j-1)k+i}$ of $\varphi(w)$. Now, for all $k \geq 2$, we define the language

$$L_k = \{\, uz_1\$^i z_2 v \mid j \geq 1, u \in \{a, b\}^{j \cdot k-1}, z_1, z_2 \in \{\bar{a}, \bar{b}\}, 1 \leq i \leq k, v \in \{a, b\}^*, \eta(v_i^{(k)}(\varphi(uz_1))) = \eta(\varphi(z_2 v)^R) \,\}.$$

The basic idea of the construction of L_k is as follows. Each word of L_k has a prefix uz_1 that contains exactly k scattered factors of the same length $j \geq 1$. The homomorphic image under φ of each factor is a binary number. Then, by $\$^i$ the ith of these binary number is addressed. Finally, the reversal of the binary number obtained from the suffix $z_2 v$ in the same way has to match the addressed binary number from the prefix. By using scattered factors, it is ensured that all these binary numbers are represented by factors of the same length. The symbols z_1 and z_2 are essentially marked digits to support reversible computations.

Proposition 3.5. *Let $k \geq 2$ be an integer. The language L_k is accepted by a* REV-DCA($k + 1$) *with time complexity $O(2^{\frac{n}{k}})$.*

Proof. We construct the REV-DCA($k + 1$) $M = \langle Q, \{a, b, \bar{a}, \bar{b}, \$\}, k, \rhd, \lhd, \delta, q_0, F\rangle$ accepting L_k by setting

$$Q = \{q_0, r_0\} \cup \{\, q_i, q_i^{(1)}, q_i^{(2)}, q_i^{(3)}, p_i, r_i, s_i, s_i^{(1)}, s_i^{(2)}, s_i^{(3)} \mid 1 \leq i \leq k \,\},$$

$F = \{\, p_i \mid 1 \leq i \leq k \,\}$, and defining δ as follows.

According to the structure of inputs from L_k the REV-DCA$(k+1)$ M works in three phases. In the first phase the input prefix uz_1 is processed. The goal of the phase is to store $\eta(v_i^{(k)}(\varphi(uz_1)))$ into counter i, for $1 \leq i \leq k$. To this end, the idea is to process the first k input symbols of uz_1 by storing their values suitably in the counters 1 to k by using the auxiliary counter $k+1$. Then the next k inputs are processed by updating the counters suitably. This cycle ends, when the barred symbol z_1 appears in the input. Let $x \in \{a, b\}$ and $d_1, d_2, \ldots, d_k \in \{+, \perp\}$ be arbitrary.

Assume for a moment that the input head of M is located on an input symbol from $\{a, b\}$ at position $\ell \cdot k + i$, while the auxiliary counter is empty and M is in state q_i, for $1 \leq i \leq k$. The next stationary transitions first store the double value of the ith counter into the empty auxiliary counter whereby the ith counter is emptied.

(1) $\delta(q_i, x, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (q_i^{(1)}, 0, 0, \ldots, 0, 1)$

(2) $\delta(q_i, x, d_1, \ldots, d_{i-1}, \perp, d_{i+1}, \ldots, d_k, d_{k+1}) = (q_i^{(2)}, 0, 0, \ldots, 0, 0)$

(3) $\delta(q_i^{(1)}, x, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (q_i, 0, 0, \ldots, 0, -1, 0, \ldots, 0, 1)$

Next, the value of the auxiliary counter is stored in the empty ith counter, whereby the auxiliary counter is emptied.

(4) $\delta(q_i^{(2)}, x, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, +) = (q_i^{(2)}, 0, 0, \ldots, 0, 1, 0, \ldots, 0, -1)$

Now, the current input symbol is processed in a non-stationary move, whereby the ith counter is incremented if it is a b and its value is kept if it is an a. Subsequently, the input head of M is at position $\ell \cdot k + i + 1$ and M is in state q', where $q' = q_{i+1}$ if $i < k$, and $q' = q_1$ if $i = k$.

(5) $\delta(q_i^{(2)}, a, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (q_i^{(3)}, 1, 0, 0, \ldots, 0)$

(6) $\delta(q_i^{(2)}, b, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (q_i^{(3)}, 1, 0, \ldots, 0, 1, 0, \ldots, 0, 0)$

(7) $\delta(q_i^{(3)}, x, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (q', 0, 0, \ldots, 0)$

The construction of δ for the first phase has to be extended for the case that a symbol from $\{\bar{a}, \bar{b}\}$ is read in state $q_k^{(2)}$. In this case, the first phase ends. So, the symbols are processed by adding one or nothing to counter k, and changing to state r_0.

(8) $\delta(q_k^{(2)}, \bar{a}, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (r_0, 1, 0, 0, \ldots, 0)$

(9) $\delta(q_k^{(2)}, \bar{b}, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (r_0, 1, 0, \ldots, 0, 1, 0)$

Finally, the very first step is done by the following transition.

(10) $\delta(q_0, \rhd, \perp, \ldots, \perp) = (q_k^{(3)}, 1, 0, 0, \ldots, 0)$

It is evident by the construction that after the first phase of an accepting computation, M is in state r_0, its auxiliary counter is empty, and the value of counter i is $\eta(v_i^{(k)}(\varphi(uz_1)))$. Moreover, in order to reach state r_0, the length of the input prefix uz_1 must be a positive multiple of k. Before we continue the construction with the second phase, we consider the reversibility of M.

A first and essential observation is that there is no state which is entered by a stationary and non-stationary transition. So, in order to determine the predecessor configuration, the state determines how to move the input head and, thus, which symbol has to be read.

Let us first consider the transitions state by state. State q_i is reached by Transitions (3) and (7). After applying Transition (3), the auxiliary counter $k+1$ is not empty, while after applying Transition (7) it must be empty. So, the predecessor configuration of some configuration with state q_i is uniquely obtained by $\delta^{\leftarrow}(q_i, x, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, +) = (q_i^{(1)}, 0, 0, \ldots, 0, 1, 0, \ldots, 0, -1)$ and $\delta^{\leftarrow}(q_i, x, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \perp) = (q', 0, 0, \ldots, 0)$, where $q' = q_{i-1}^{(3)}$ if $1 < i \leq k$, and $q' = q_k^{(3)}$ if $i = 1$.

State $q_i^{(1)}$ is only reached by Transition (1), and the predecessor configurations are uniquely obtained by $\delta^{\leftarrow}(q_i^{(1)}, x, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, +) = (q_i, 0, 0, \ldots, 0, -1)$.

State $q_i^{(2)}$ is reached by Transitions (2) and (4). After applying Transition (2), the counter i is empty, while after applying Transition (4) it must be non-empty. So, the predecessor configuration of some configuration with state $q_i^{(2)}$ is uniquely obtained by $\delta^{\leftarrow}(q_i^{(2)}, x, d_1, \ldots, d_{i-1}, \perp, d_{i+1}, \ldots, d_k, d_{k+1}) = (q_i, 0, 0, \ldots, 0)$ and $\delta^{\leftarrow}(q_i^{(2)}, x, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (q_i^{(2)}, 0, 0, \ldots, 0, -1, 0, \ldots, 0, 1)$.

State $q_i^{(3)}$ is reached by Transitions (5) and (6), and for $i = k$ also by Transition (10). These transitions are non-stationary and can be distinguished by the input symbol read. Therefore, the predecessor configuration of some configuration with state $q_i^{(3)}$ is uniquely obtained by the reversed transitions

$$\begin{aligned}\delta^{\leftarrow}(q_i^{(3)}, a, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \bot) &= (q_i^{(2)}, -1, 0, \ldots, 0),\\ \delta^{\leftarrow}(q_i^{(3)}, b, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, \bot) &= (q_i^{(2)}, -1, 0, \ldots, 0, -1, 0, \ldots, 0, 0),\\ \delta^{\leftarrow}(q_k^{(3)}, \rhd, \bot, \ldots, \bot) &= (q_0, -1, 0, \ldots, 0).\end{aligned}$$

So, we conclude that the construction of the first phase is reversible. In the second phase, M reads at least one and at most k symbols \$. The number of \$'s selects the counter to which the number represented by the mirrored suffix $(z_2v)^R$ has to be compared.

$$\begin{aligned}&(11)\quad \delta(r_{i-1}, \$, d_1, \ldots, d_k, \bot) = (r_i, 1, 0, \ldots, 0)\\ &(12)\quad \delta(r_i, \bar{a}, d_1, \ldots, d_k, \bot) = (s_i, 1, 0, \ldots, 0)\\ &(13)\quad \delta(r_i, \bar{b}, d_1, \ldots, d_k, \bot) = (s_i, 1, 0, \ldots, 0, -1, 0, \ldots, 0)\end{aligned}$$

Since $1 \leq i \leq k$, at least one and at most k symbols \$ can be read. By construction, after the second phase of an accepting computation, M is in state s_i, where the factor $\$^i z_2$ has been processed. In order to reach state s_i by Transition (12) or (13) the auxiliary counter must be empty. As above, before we continue the construction with the third phase, we consider the reversibility of M in the second phase.

State r_0 is reached by Transitions (8) and (9). These transitions are non-stationary and can be distinguished by the input symbol read. Therefore, the predecessor configuration of some configuration with state r_0 is uniquely obtained by the reversed transitions $\delta^{\leftarrow}(r_0, \bar{a}, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \bot) = (q_k^{(2)}, -1, 0, \ldots, 0)$ and $\delta^{\leftarrow}(r_0, \bar{b}, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \bot) = (q_k^{(2)}, -1, 0, \ldots, -1, 0)$.

For $1 \leq i \leq k$, state r_i is only reached by Transition (11), and the predecessor configurations are uniquely obtained by $\delta^{\leftarrow}(r_i, \$, d_1, \ldots, d_k, \bot) = (r_{i-1}, 0, 0, \ldots, 0)$. State s_i is dealt with below.

Finally, in the third phase the value of counter i is compared with the number $\eta(\varphi(z_2v)^R)$. To this end, after reading some input symbol, the ith counter is decremented for a b and kept as it is for an a. Subsequently, by using the auxiliary counter $k+1$, the value of counter i is divided by two. If it is odd, the comparison failed and the computation halts rejecting. This process is repeated until the value of counter i is zero. If in this situation the right endmarker appears in the input, the values match and M accepts in state p_i. Let $y \in \{a, b, \lhd\}$ be arbitrary. The next transitions divide the value of counter i by two and store the result in the auxiliary counter, whereby counter i is emptied.

$$\begin{aligned}&(14)\quad \delta(s_i, y, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \bot) = (s_i^{(1)}, 0, 0, \ldots, 0)\\ &(15)\quad \delta(s_i^{(1)}, y, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (s_i^{(2)}, 0, 0, \ldots, 0, -1, 0, \ldots, 0)\\ &(16)\quad \delta(s_i^{(1)}, y, d_1, \ldots, d_{i-1}, \bot, d_{i+1}, \ldots, d_k, d_{k+1}) = (s_i^{(3)}, 0, 0, \ldots, 0)\\ &(17)\quad \delta(s_i^{(2)}, y, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (s_i^{(1)}, 0, 0, \ldots, 0, -1, 0, \ldots, 0, 1)\end{aligned}$$

Next, the value of the auxiliary counter is stored in the empty ith counter, whereby the auxiliary counter is emptied. Subsequently, the entire process is repeated by processing the next input symbol.

$$\begin{aligned}&(18)\quad \delta(s_i^{(3)}, y, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, +) = (s_i^{(3)}, 0, 0, \ldots, 0, 1, 0, \ldots, 0, -1)\\ &(19)\quad \delta(s_i^{(3)}, a, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, \bot) = (s_i, 1, 0, \ldots, 0)\\ &(20)\quad \delta(s_i^{(3)}, b, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, \bot) = (s_i, 1, 0, \ldots, 0, -1, 0, \ldots, 0)\end{aligned}$$

The repetition ends, if the ith counter is empty when the next input symbol is to be read. In this case, the right endmarker has to be read for acceptance.

$$(21)\quad \delta(s_i^{(3)}, \lhd, d_1, \ldots, d_{i-1}, \bot, d_{i+1}, \ldots, d_k, \bot) = (p_i, 1, 0, \ldots, 0)$$

Clearly, M accepts if and only if the input belongs to L_k. Let us consider the reversibility of the third phase.

State s_i is reached by Transitions (12), (13), (19), and (20). These transitions are non-stationary and can be distinguished by the input symbol read. Therefore, the predecessor configuration of some configuration with state s_i is uniquely obtained by the reversed transitions

$$\begin{aligned}
\delta^{\leftarrow}(s_i, \bar{a}, d_1, \ldots, d_k, \bot) &= (r_i, -1, 0, \ldots, 0),\\
\delta^{\leftarrow}(s_i, \bar{b}, d_1, \ldots, d_k, \bot) &= (r_i, -1, 0, \ldots, 0, 1, 0, \ldots, 0),\\
\delta^{\leftarrow}(s_i, a, d_1, \ldots, d_k, \bot) &= (s_i^{(3)}, -1, 0, \ldots, 0), \text{ and}\\
\delta^{\leftarrow}(s_i, b, d_1, \ldots, d_k, \bot) &= (s_i^{(3)}, -1, 0, \ldots, 0, 1, 0, \ldots, 0).
\end{aligned}$$

State $s_i^{(1)}$ is reached by Transitions (14) and (17). After applying Transition (14), the auxiliary counter is empty, while after applying Transition (17) it must be non-empty. So, the predecessor configuration of some configuration with state $s_i^{(1)}$ is uniquely obtained by $\delta^{\leftarrow}(s_i^{(1)}, y, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, \bot) = (s_i, 0, 0, \ldots, 0)$ and $\delta^{\leftarrow}(s_i^{(1)}, y, d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_k, +) = (s_i^{(2)}, 0, 0, \ldots, 0, 1, 0, \ldots, 0, -1)$.

State $s_i^{(2)}$ is only reached by Transition (15), and the predecessor configurations are uniquely obtained by $\delta^{\leftarrow}(s_i^{(2)}, y, d_1, \ldots, d_{k+1}) = (s_i^{(1)}, 0, 0, \ldots, 0, 1, 0, \ldots, 0)$.

State $s_i^{(3)}$ is reached by Transitions (16) and (18). After applying Transition (16), the ith counter is empty, while after applying Transition (18) it must be non-empty. So, the predecessor configuration of some configuration with state $s_i^{(3)}$ is uniquely obtained by $\delta^{\leftarrow}(s_i^{(3)}, y, d_1, \ldots, d_{i-1}, \bot, d_{i+1}, \ldots, d_k, d_{k+1}) = (s_i^{(1)}, 0, 0, \ldots, 0)$ and $\delta^{\leftarrow}(s_i^{(3)}, y, d_1, \ldots, d_{i-1}, +, d_{i+1}, \ldots, d_k, d_{k+1}) = (s_i^{(3)}, 0, 0, \ldots, 0, -1, 0, \ldots, 0, 1)$.

Finally, the accepting state p_i is only reached by Transition (21), and the predecessor configurations are uniquely obtained by $\delta^{\leftarrow}(p_i, \lhd, d_1, \ldots, d_{i-1}, \bot, d_{i+1}, \ldots, d_k, \bot) = (s_i^{(3)}, -1, 0, \ldots, 0)$.

So, we have that M is reversible and accepts L_k. It remains to consider its time complexity. For the length of each word $w = uz_1\$^i z_2 v$ belonging to $L(M)$ we have $|w| \geq j \cdot k + 1 + 1 \geq j \cdot k$, where $j \geq 1$. Note, that due to transition (19) no 'leading zeroes' are possible when representing a number by the suffix $z_2 v$.

When M processes w, in the first phase the values $\eta(v_i^{(k)}(\varphi(uz_1)))$, $1 \leq i \leq k$, are stored into the counters 1 to k. For each $\eta(v_i^{(k)}(\varphi(uz_1)))$ this takes at most $2 \cdot 2^j + O(j) \in O(2^j)$ time steps. Then the \$'s are read which takes another at most k time steps. Finally, processing the suffix takes another $O(2^j)$ time steps. In total, these are at most $(k+1)O(2^j) + k = O(2^j)$ steps. Since $n = |w| \geq j \cdot k$, we conclude that M obeys the time complexity $O(2^{\frac{n}{k}})$. □

In order to prove that k counters are not enough to accept L_k in time $O(2^{\frac{n}{k}})$, we use again Kolmogorov complexity and incompressibility arguments. In particular, k counters are not enough even for not necessarily reversible counter automata.

Proposition 3.6. *Let $k \geq 2$ be an integer. The language L_k is not accepted by any* DCA(k) *with time complexity $O(2^{\frac{n}{k}})$.*

Proof. We assume in contrast to the assertion that L_k is accepted by some DCA(k) $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F\rangle$ with time complexity $O(2^{\frac{n}{k}})$.

We choose some integer $j \geq 1$ large enough and a word $u' \in \{a, b\}^{j \cdot k}$ such that $C(u') \geq |u'|$. Next, we consider the computation of M on the prefix uz_1, where uz_1 is essentially u' but with the last symbol barred. In particular, we consider the configuration reached exactly after M has moved its input head from z_1 to the right.

Set $\ell = \frac{n}{k}$. Since M accepts in time $O(2^{\frac{n}{k}})$ the maximum number stored in some counter of M in the configuration is at most $c \cdot 2^\ell$, for some constant $c \geq 0$. Moreover, in [17] it is shown that in an accepting computation if a counter machine has $|Q|$ states then immediately after reading a prefix u' of its input the value of at least one counter is bounded from above by $(|Q|+1) \cdot |u'|$. Since this could be each of the k counters, the configuration in question without its second and third component can be encoded with

$$\begin{aligned}
\lceil \log(|Q| \cdot (c \cdot 2^\ell)^{k-1} \cdot k \cdot (|Q|+1) \cdot |u'|) \rceil &\leq c' + \ell(k-1) + \log(|u'|)\\
&= c' + \frac{n}{k}(k-1) + \log(|u'|)
\end{aligned}$$

bits, for some constant $c' \geq 0$. Since $n \leq j \cdot k + j + k$, we have $\frac{n-k}{k+1}k \leq j \cdot k = |u'|$ and conclude

$$\begin{aligned} c' + \frac{n}{k}(k-1) + \log(|u'|) &\leq c' + \left(\frac{|u'|(k+1)}{k} + k\right)\frac{k-1}{k} + \log(|u'|) \\ &= c' + \frac{|u'|(k+1)(k-1)}{k^2} + \frac{(k-1)k}{k} + \log(|u'|) \\ &= c' + |u'| - \frac{|u'|}{k^2} + k - 1 + \log(|u'|) \\ &< |u'|. \end{aligned}$$

Knowing M, the length of u', and the configuration in question without its second and third component, u' can be reconstructed as follows. In order to reconstruct u' it is sufficient to reconstruct the k scattered factors $v_i^{(k)}(\varphi(u'))$. For each of these factors, we test all non-empty candidates from $\{a,b\}^{\leq \frac{|u'|}{k}}$. A candidate v is tested by replacing its first symbol by the barred version, preceding it by $\$^i$, and feeding it to M. Now it is sufficient to simulate M on this input starting in the encoded configuration. The simulation can be made halting due to the upper bound of the time complexity. So, if M accepts then v^R preceded with leading a's to obtain the length $\frac{|u'|}{k}$ gives the scattered factor $v_i^{(k)}(\varphi(u'))$. Moreover, there is an accepted candidate for each i. Therefore, the reconstruction terminates.

We conclude that the Kolmogorov complexity of u' is bounded from above by a positive constant which gives the size of M and the program that reconstructs u' plus $\log(|u'|)$ plus the size of the encoding of the configuration. In total this is strictly less than $|u'|$. However, this is a contradiction since u' has been chosen such that $C(u') \geq |u'|$. The contradiction shows that L_k is not accepted by M with time complexity $O(2^{\frac{n}{k}})$. □

So, we have a strict counter hierarchy for reversible as well as irreversible counter automata obeying the exponential time complexity $O(2^{\frac{n}{k}})$. To compare reversible counter automata with some $k+1$ counters to irreversible counter automata with k counters, we know from above that language L_k is accepted by the former but is not accepted by the latter devices. But what about the other direction? We cannot utilize the regular language provided by Theorem 3.3 for this purpose. It is not accepted by any $2^{o(n)}$-time REV-DCA$(k+1)$ but here we are considering the exponential time complexity $O(2^{\frac{n}{k}})$. Therefore, the question remains open. So, we turn to consider counter hierarchies for polynomial time complexities. Such hierarchies are known to exist for irreversible counter automata that have to respect the time complexity on all inputs, that is, also on inputs *not accepted* [17]. Again, here we improve the result that witness languages require a certain number of counters such that only accepting computations have to respect the time complexity. To this end, we use once more Kolmogorov arguments. The witness languages are modifications of the languages L_k used in the proofs of Proposition 3.5 and Proposition 3.6.

For all $k \geq 2$ and $s \geq 1$, we define the language

$$\begin{aligned} \tilde{L}_{k,s} = \{\, b^k\#uz_1\$^i z_2 v z_3 v' \# a^{2^j} \mid\ & j \geq 1, b^k u \in \{a,b\}^{j\cdot k\cdot s-1}, \\ & z_1, z_2, z_3 \in \{\bar{a},\bar{b}\}, 1 \leq i \leq k-1, v, v' \in \{a,b\}^*, \\ & \eta(v_{i+1}^{(k)}(\varphi(b^k u z_1))) = \eta(\varphi(z_2 v)^R), \eta(v_1^{(k)}(\varphi(b^k u z_1))) = \eta(\varphi(z_3 v')^R) \,\}. \end{aligned}$$

Proposition 3.7. *Let $k \geq 2$ and $s \geq 1$ be integers. The language $\tilde{L}_{k,s}$ is accepted by a* REV-DCA$(k+1)$ *with time complexity $O(n^s)$.*

Proof. We modify the construction of the REV-DCA$(k+1)$ M from the proof of Proposition 3.5 that accepts the language L_k. The resulting REV-DCA$(k+1)$ M' accepts $\tilde{L}_{k,s}$ with time complexity $O(n^s)$.

Essentially, in a first phase M' processes an input prefix $b^k\#uz_1$ exactly as M processes $b^k uz_1$ in its first phase. The only differences are that M' uses additional states to verify that the prefix begins with b^k which can

be done reversibly due to the separating symbol #, and that M' uses further new states to check that $|b^k u z_1|$ is a multiple of $k \cdot s$ instead of a multiple of k. So, at the end of this phase, counter i has the value $\eta(v_i^{(k)}(\varphi(b^k u z_1)))$, $1 \leq i \leq k$, and the auxiliary counter is empty. Moreover, the length of the input prefix $b^k u z_1$ must be a positive multiple of $k \cdot s$.

In the second phase, M' reads at least one and at most $k-1$ symbols \$. The number of \$'s selects the counter to which the number represented by the mirrored factor $(z_2 v)^R$ has to be compared. The second phase of M' corresponds to the second phase of M except that the $(i+1)$st counter is addressed instead of the ith counter. After the second phase of an accepting computation, M' has processed the factor $\$^i z_2$.

Next, M' performs its third phase in the same way as M performs its third phase. That is, the value of counter $i+1$ is compared with the number $\eta(\varphi(z_2 v)^R)$. At the end of the third phase of an accepting computation, counter $i+1$ as well as the auxiliary counter are empty, and M' has processed the factor $v z_3$.

Now M' starts its fourth phase. It is similar to the third one, but compares the value stored in the first counter with the number $\eta(\varphi(z_3 v')^R)$. In addition, counter $i+1$ is incremented by one for each s symbols read by M' from $z_3 v'$. Let us consider the numbers stored in the counters in more detail. Since the input begins with b^k, every scattered subword $v_i^{(k)}(\varphi(b^k u z_1))$ begins with 1. Moreover, it is of length $j \cdot s$. Therefore, $\eta(v_i^{(k)}(\varphi(b^k u z_1)))$ is of order $\Theta(2^{j \cdot s})$. In particular, we have that $\eta(\varphi(z_3 v')^R)$ is of order $\Theta(2^{j \cdot s})$. At the end of the fourth phase of an accepting computation, counter 1 and the auxiliary counter are empty, counter $i+1$ stores the value j, and M' has processed the factor v'#.

In its final fifth phase, M' first increments counter 1 to value 1 which is clearly reversible. Then it doubles the value of counter 1 in the same way as has been shown in the construction of the proof of Proposition 3.5 (transitions (1), (2), (3)). To this end, it uses the auxiliary counter. After the doubling, the value j of counter $i+1$ is decreased by one. This doubling is repeated until counter $i+1$ is empty, that is, j times. In the end of this process, counter $i+1$ and the auxiliary counter are empty while counter 1 has the value 2^j. This phase is reversible, since each doubling is reversible as shown in the proof of Proposition 3.5. An intuition of the reverse process is to divide the value of counter 1 by two, again with the help of the auxiliary counter, whereby the value of counter $i+1$ is increased by one. This division is reversible as shown in the proof of Proposition 3.5 (transitions (14), (15), (16), (17)). Now the process of division is repeated until the result becomes 1, that is, j times. In the end of this process, counter $i+1$ has the value j, the auxiliary counter is empty, and counter 1 has the value 1.

In the next phase of the forward processing of M', the remaining input suffix a^{2^j} is matched with the value of counter 1. The input is accepted if counter 1 gets empty when the endmarker appears in the input. This is also reversible.

So, M' accepts $\tilde{L}_{k,s}$ and it is not hard to see that M' is reversible since M is and we included the separating symbols z_3 and #.

We turn to the time complexity of M'. Consider some word

$$w = b^k \# u z_1 \$^i z_2 v z_3 v' \# a^{2^j}$$

belonging to $L(M') = \tilde{L}_{k,s}$. From above we know that $v_i^{(k)}(\varphi(b^k u z_1))$ is of length $j \cdot s$ and $\eta(v_i^{(k)}(\varphi(b^k u z_1)))$ is of order $\Theta(2^{j \cdot s})$. So, for the length of w we have $n = |w| \in \Theta(j \cdot s \cdot k + 1 + i + j \cdot s \cdot 2 + 1 + 2^j) = \Theta(2^j)$ Note, that as above no 'leading zeroes' are possible when representing a number by the suffixes $z_2 v$ and $z_3 v'$. We conclude that $n^s \in \Theta(2^{j \cdot s})$.

When M processes w, the first three phases correspond to the three phases in the proof of Proposition 3.5. That is, they take $O(2^{j \cdot s})$ time steps. The fourth phase takes another $O(2^{j \cdot s})$ steps and the fifth phase $O(2^j)$ further steps. In total these are $O(2^{j \cdot s})$ steps and, thus, M' accepts in $O(n^s)$ time. □

In order to prove that k counters are not enough to accept $\tilde{L}_{k,s}$ in time $O(n^s)$, we proceed along the lines of the proof of Proposition 3.6 and apply incompressibility arguments. Also here we will derive that k counters are not enough even for not necessarily reversible counter automata.

Proposition 3.8. *Let $k \geq 2$ and $s \geq 1$ be integers. The language $\tilde{L}_{k,s}$ is not accepted by any* DCA(k) *with time complexity $O(n^s)$.*

Proof. In contrast to the assertion assume that L_k is accepted by some DCA(k) $M = \langle Q, \Sigma, k, \rhd, \lhd, \delta, q_0, F\rangle$ with time complexity $O(n^s)$.

We choose some integer $j \geq 1$ large enough and a word $u' \in \{a,b\}^{j \cdot s \cdot k - k}$ such that $C(u') \geq |u'|$. Next, we consider the computation of M on the prefix $b^k\#uz_1$, where uz_1 is essentially u' but with the last symbol barred. In particular, we consider the configuration reached exactly after M has moved its input head from z_1 to the right.

Since M accepts in time $O(n^s)$ the maximum number stored in some counter of M in the configuration is at most $c \cdot n^s$, for some constant $c \geq 0$. From the proof of Proposition 3.7 we derive $n = \Theta(2^j)$ and, thus, the maximum number stored in some counter of M in the configuration is at most $c \cdot 2^{j \cdot s}$, for some constant $c \geq 0$. In [17] it is shown that in an accepting computation if a counter machine has $|Q|$ states then immediately after reading a prefix u' of its input the value of at least one counter is bounded from above by $(|Q|+1) \cdot |u'|$. Since this could be each of the k counters, the configuration in question without its second and third component can be encoded with

$$\lceil \log(|Q| \cdot (c \cdot 2^{j \cdot s})^{k-1} \cdot k \cdot (|Q|+1) \cdot (k+1+|u'|)) \rceil \leq c' + j \cdot s \cdot (k-1) + \log(|u'|)$$

bits, for some constant $c' \geq 0$.

Clearly, with the knowledge of M, the number j, and the configuration in question without its second and third component, the k scattered factors $v_i^{(k)}(\varphi(b^k u'))$ and, thus, u' can be reconstructed as follows. Each pair $z_2 v z_3 v'$ from

$$\{\bar{a}, \bar{b}\}\{a,b\}^{\leq j \cdot s - 1}\{\bar{a}, \bar{b}\}\{a,b\}^{\leq j \cdot s - 1}$$

is tested by feeding $\$^i z_2 v z_3 v' \# a^{2^j}$ to M, for all $1 \leq i \leq k-1$. A pair is tested by simulating M starting in the encoded configuration. The simulation can be made halting due to the upper bound of the time complexity. So, if M accepts then the tested pair tells us $v_1^{(k)}(\varphi(b^k u'))$ and $v_i^{(k)}(\varphi(b^k u'))$. Moreover, there is an accepted pair for each i. Therefore, the reconstruction terminates.

We conclude that the Kolmogorov complexity of u' is bounded from above by a positive constant c'' which gives the size of M and the program that reconstructs u' plus $\log(j)$ plus the size of the encoding of the configuration. In total this is at most

$$\begin{aligned} &c'' + \log(j) + c' + j \cdot s \cdot (k-1) + \log(|u'|) \\ &= c''' + \log(j) + j \cdot s \cdot k - j \cdot s + \log(j \cdot s \cdot k - k) \\ &< j \cdot s \cdot k - k = |u'| \end{aligned}$$

for some positive constant c''' and j large enough. However, this is a contradiction since u' has been chosen such that $C(u') \geq |u'|$. The contradiction shows that $\tilde{L}_{k,s}$ is not accepted by M with time complexity $O(n^s)$. □

Besides the infinite and strict counter hierarchies for reversible and general counter automata, Proposition 3.7 and Proposition 3.8 complement Theorem 3.3 in the sense that $k+1$ reversible counters are not strictly weaker than k general counters and vice versa. In particular, we have the following incomparabilities (see Fig. 1 for a diagram showing the relations between language families).

Theorem 3.9. *Let $k_1 > k_2 \geq 0$ and $s \geq 1$ be integers. The family of languages accepted by* REV-DCA(k_1) *in $O(n^s)$ time is incomparable with the family of languages accepted by* DCA(k_2) *in $O(n^s)$ time.*

Proof. By Theorem 3.3 there is a regular language not belonging to any of the families $\mathscr{L}$(REV-DCA(k_1)). Clearly, all regular languages are accepted by some DCA(0) in real time.

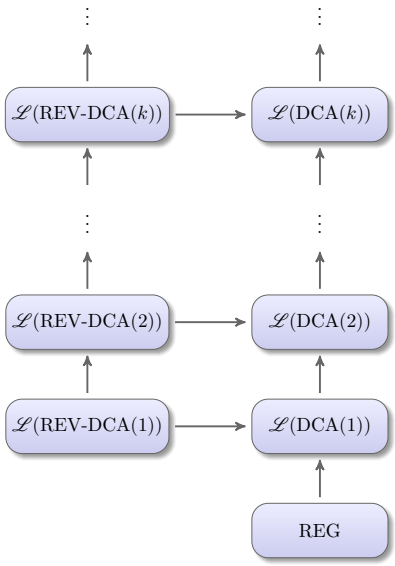

FIGURE 1. Relationships between language families induced by counter automata working in $O(n^s)$ time. An arrow between families indicates a strict inclusion. Whenever two families are not connected by a path they are incomparable.

For the other direction, Example 2.2 provides a deterministic context-free language accepted by some REV-DCA(1) in real time. Since the language is not regular, it cannot be accepted by any DCA(0). Moreover, Example 2.3 provides a non-context-free language accepted by some REV-DCA(2) in real time. Since the language is not context-free, it cannot be accepted by any DCA(1).

Finally, for $k_1 \geq 2$, Proposition 3.7 shows that there is a language accepted by some REV-DCA(k_1) in time $O(n^s)$ that is not accepted by any $O(n^s)$-time DCA(k_2) by Proposition 3.8. □

4. Decidability problems

In this section, we study decidability problems for real-time REV-DCA(k) with $k \geq 1$. In case of machines with at least two counters, it turns out that all usually studied questions such as emptiness, finiteness, inclusion, equivalence, or regularity are not semidecidable. We recall (see, for example, [23]) that a decidability problem is *semidecidable* (*decidable*) if and only if the set of all instances for which the answer is 'yes' is recursively enumerable (recursive). Clearly, any decidable problem is also semidecidable, while the converse does not generally hold.

In case of machines with one counter, it turns out that almost all questions are decidable, since real-time REV-DCA(1) can be considered as reversible deterministic pushdown automata and the class has many decidable questions. However, the inclusion problem is undecidable for DPDAs and we will show here that it is even not semidecidable for real-time REV-DCA(1).

The non-semidecidability results are shown by reduction of the emptiness problem of Turing machines. It is well known that emptiness for such machines is not semidecidable (see, for example, [23]).

The technique to obtain this reduction is based on the notion of *valid computations of multiplying counter machines* where we are following an idea and the notation given in [24]. Valid computations of a multiplying counter machine are, basically, histories of multiplying counter machine computations which are encoded into single words. It will be shown that the set of such suitably formatted valid computations can be represented as the intersection of two languages accepted by real-time REV-DCA(1). Since it is known [24] that every language accepted by a Turing machine can be accepted, suitably encoded, by some multiplying counter machine as well,

the emptiness of a Turing machine can be reduced to the emptiness of the intersection of two languages accepted by real-time REV-DCA(1). From this the non-semidecidability of inclusion for real-time REV-DCA(1) and the non-semidecidability results for real-time REV-DCA(k) with $k \geq 2$ can be derived.

We start by summarizing the necessary notations given in [24]. A *multiplying counter machine* is a one-register machine (the register is capable of holding an arbitrary integer) which can multiply the content of its register by one of a finite number of multiplicands, and branching if the resulting product is not an integer. Formally, let Q be a finite set of states with two distinguished elements $q_0 \neq q_f$, called the initial and final state. Let $C = \{2, 3, 5, 7, 1/2, 1/3, 1/5, 1/7\}$ be a finite set of multiplicands. A transition rule is an element of $Q \times C \times Q \times Q$. A multiplying counter machine M is a set of transition rules such that no two transition rules have the same first component and no transition rule of M has q_f as the first component nor q_0 as the last component. A configuration of M is string of the form qa^i, where $q \in Q$. For each integer n, we write $qa^n \vdash pa^{kn}$, if (q, k, p, r) is a transition rule of M and kn is an integer. We write $qa^n \vdash ra^n$, if (q, k, p, r) is a transition rule of M and kn is not an integer. A valid computation is a string built from a sequence of configurations passed through during a computation starting in configuration $q_0a^{2^i}$ for some $i \geq 0$ and halting in state q_f.

Theorem 4.1 [24]**.** *Let $L \subseteq \{1, 2\}^*$ be a set of strings accepted by a Turing machine and let each string $x_1x_2 \cdots x_n \in \{1, 2\}^*$ be encoded as a natural number $i = x_1 + 3x_2 + 3^2x_3 + \cdots + 3^{n-1}x_n$. Then, a multiplying counter machine M can effectively be constructed such that $q_0a^{2^i} \vdash^* q_fa^j$ (for some $j \geq 0$) if and only if $i = x_1 + 3x_2 + 3^2x_3 + \cdots + 3^{n-1}x_n$ and $x_1x_2 \cdots x_n$ is in L.*

Now, a valid computation of a multiplying counter machine M (VALC(M)) is defined in [24] as a string

$$q_0aq_0a^2q_0a^4 \cdots q_0a^{2^{i-1}}\alpha_0\alpha_1 \cdots \alpha_n$$

where $\alpha_0, \alpha_1, \ldots, \alpha_n$ are configurations of M such that $\alpha_0 = q_0a^{2^i}$, $\alpha_n = q_fa^p$ for some $p \geq 1$, and $\alpha_j \vdash \alpha_{j+1}$ for $0 \leq j < n$.

Our goal is to represent the set of valid computations as the intersection of two languages where each of which is accepted by a real-time *reversible* one-counter automaton. Hence, we consider several modifications to the set of valid computations to enable a reversible computation. First, we assume that $n + i + 1$ is even, that is, every valid computation consists of an even number of configurations. This is due to technical reasons and facilitates later constructions. If $n + i + 1$ is odd, then we consider a new state $q_{f'}$, replace $\alpha_n = q_fa^p$ by $\alpha_n = q_{f'}a^p$, add a configuration $\alpha_{n+1} = q_fa^p$, and obtain an even number of configurations.

Second, we add the state of each configuration to the end of a configuration. That is, $\alpha_j = q_ja^{n_j}$ is modified to $\alpha_jq_j = q_ja^{n_j}q_j$ and for the initial phase of doubling a-blocks we modify $q_0a^{2^p}$ $(p \geq 0)$ to $q_0'a^{2^p}q_0'$, where q_0' is a new state not in Q.

The third modification concerns configurations $qa^n \vdash pa^{kn}$, if (q, k, p, r) is a transition rule of M and kn is an integer. In this case, we store the information of the multiple k as superscript (k) in the state p of the successor configuration. In addition, we store in state q as subscript the result of n mod $(1/k)$, if $k < 1$. That is, $qa^nqpa^{kn}p$ is modified to $qa^nq_{(n \bmod (1/k))}p^{(k)}a^{kn}p^{(k)}$ and $q_0'a^{2^{i-1}}q_0'q_0a^{2^i}q_0$ is modified to $q_0'a^{2^{i-1}}q_0'q_0^{(2)}a^{2^i}q_0^{(2)}$.

The fourth modification concerns configurations $qa^n \vdash ra^n$, if (q, k, p, r) is a transition rule of M and kn is not an integer. Hence, $k \in \{1/2, 1/3, 1/5, 1/7\}$ and we store the information (1) as superscript in the state r of the successor configuration. In addition, we store in state q as subscript the result of n mod $(1/k)$. That is, qa^nqra^nr is modified to $qa^nq_{(n \bmod (1/k))}r^{(1)}a^nr^{(1)}$.

If the state $q_{f'}$ has been introduced in the first modification, we modify as fifth modification $q_{f'}a^pq_{f'}q_fa^pq_f$ to $q_{f'}a^pq_{f'}q_f^{(1)}a^pq_f^{(1)}$.

Finally, we mark the a in the first block, that is, $q_0'a$ is replaced by $q_0'a'$ and $q_0^{(\ell_0)}a$ is replaced by $q_0^{(\ell_0)}a'$ for $\ell_0 = 2$.

Formally, the set VALC$'(M)$ of valid computations of a multiplying counter machine M is the following set of strings. Let $i = x_1 + 3x_2 + 3^2x_3 + \cdots + 3^{n-1}x_n$ and a^{2^i} be the unary encoding of a string $x_1x_2 \cdots x_n \in L$. If

$i > 0$, we have

$$q_0' a' q_0' q_0' a^2 q_0' \cdots q_0' a^{2^{i-1}} q_0' q_0^{(\ell_0)} a^{n_0} q_{0,\varphi_0}^{(\ell_0)} q_1^{(\ell_1)} a^{n_1} q_{1,\varphi_1}^{(\ell_1)} \cdots \cdots q_j^{(\ell_j)} a^{n_j} q_{j,\varphi_j}^{(\ell_j)} q_{j+1}^{(\ell_{j+1})} a^{n_{j+1}} q_{j+1,\varphi_{j+1}}^{(\ell_{j+1})} \cdots q_f^{(\ell_n)} a^{n_n} q_f^{(\ell_n)},$$

if $i = 0$, we have

$$q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)} q_1^{(\ell_1)} a^{n_1} q_{1,\varphi_1}^{(\ell_1)} \cdots q_j^{(\ell_j)} a^{n_j} q_{j,\varphi_j}^{(\ell_j)} q_{j+1}^{(\ell_{j+1})} a^{n_{j+1}} q_{j+1,\varphi_{j+1}}^{(\ell_{j+1})} \cdots q_f^{(\ell_n)} a^{n_n} q_f^{(\ell_n)},$$

where $q_j a^{n_j}$, for $1 \leq j \leq n$, are configurations of M such that $n_0 = 2^i$ and $q_j a^{n_j} \vdash q_{j+1} a^{n_{j+1}}$, for $0 \leq j < n$. Furthermore, due to the definition of M we know that for $0 \leq j \leq n$ each state q_j different from q_f and $q_{f'}$ has an associated number $k_j \in C$. If $k_j > 1$ or, if $k_j < 1$ and n_j is divisible by $1/k_j$, we define, for $0 < j \leq n$, $\ell_j = k_{j-1}$ and $\ell_j = 1$ otherwise. Moreover, we set $\ell_0 = 2$. For $0 \leq j < n$ we define $\varphi_j = n_j \bmod (1/k_j)$, if $k_j < 1$, and $\varphi_j = 0$ otherwise. We illustrate the definition with the following example.

Example 4.2. We consider some valid computation

$$q_0 a q_0 a^2 q_0 a^4 q_0 a^8 q_0 a^{16} q_1 a^{32} q_2 a^{16} q_3 a^{16} q_4 a^8 q_f a^8$$

of a multiplying counter machine M with transitions $(q_0, 2, q_1, \cdot)$, $(q_1, \frac{1}{2}, q_2, \cdot)$, $(q_2, \frac{1}{3}, \cdot, q_3)$, $(q_3, \frac{1}{2}, q_4, \cdot)$, and $(q_4, \frac{1}{5}, \cdot, q_f)$. States not relevant for the example are denoted by $\cdot$. According to the above discussion we obtain

$$q_0' a' q_0' q_0' a^2 q_0' q_0' a^4 q_0' q_0' a^8 q_0' q_0^{(2)} a^{16} q_{0,0}^{(2)} q_1^{(2)} a^{32} q_{1,0}^{(2)} q_2^{(\frac{1}{2})} a^{16} q_{2,1}^{(\frac{1}{2})} q_3^{(1)} a^{16} q_{3,0}^{(1)} q_4^{(\frac{1}{2})} a^8 q_{4,3}^{(\frac{1}{2})} q_f^{(1)} a^8 q_f^{(1)}.$$

■

Our next goal is to represent the set $\mathrm{VALC}'(M)$ of such modified valid computations as the intersection of two languages $\mathrm{VALC}'_1(M)$ and $\mathrm{VALC}'_2(M)$ that are accepted by real-time REV-DCA(1). To this end, we define $\mathrm{VALC}'_1(M)$ to be the set of strings that start with $q_0' a' q_0'$ or $q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)}$, end with $q_f^{(k)} a^* q_f^{(k)}$ $(k \in C \cup \{1\})$, have no a' or $q_f^{(k)}$ in between, and we require that the successor configuration of any configuration at an odd position is correctly computed. Similarly, $\mathrm{VALC}'_2(M)$ is the set of strings that have the same format as $\mathrm{VALC}'_1(M)$ and the successor configuration of any configuration at an even position is correctly computed. Due to the required format of $\mathrm{VALC}'_1(M)$ and $\mathrm{VALC}'_2(M)$ we obtain that $\mathrm{VALC}'_1(M) \cap \mathrm{VALC}'_2(M) = \mathrm{VALC}'(M)$. Moreover, we have that $\mathrm{VALC}'(M)$ is empty if and only if M accepts the empty set.

Lemma 4.3. *Let M be a multiplying counter machine. Then two real-time* REV-DCA(1) *accepting the sets* $\mathrm{VALC}'_1(M)$ *and* $\mathrm{VALC}'_2(M)$ *can effectively be constructed from M.*

Proof. We describe the construction of a real-time REV-DCA(1) accepting the set $\mathrm{VALC}'_1(M)$. A real-time REV-DCA(1) accepting the set $\mathrm{VALC}'_2(M)$ can similarly be constructed. First we note that the required correct formatting of the input, namely, starting with $q_0' a' q_0'$ or $q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)}$, ending with $q_f^{(k)} a^* q_f^{(k)}$ $(k \in C \cup \{1\})$, and having no a' or $q_f^{(k)}$ in between, can be tested by a reversible deterministic finite automaton. Hence, this test can be realized in an additional component using the standard cross product construction. (see, *e.g.*, [14], where the construction for reversible pushdown automata is described).

We note that any string in $\mathrm{VALC}'_1(M)$ consists of a sequence of blocks of adjacent configurations having one of the following forms:

1. $q_0' a^n q_0' q_0' a^{2n} q_0'$ for some $n \geq 1$ (if $n = 1$, $a^n = a'$),

2. $q_0' a^n q_0' q_0^{(2)} a^{2n} q_{0,\varphi_0}^{(2)}$ for some $n \geq 1$ (if $n = 1$, $a^n = a'$),
3. $q_j^{(\ell_j)} a^{n_j} q_{j,\varphi_j}^{(\ell_j)} q_{j+1}^{(\ell_{j+1})} a^{n_{j+1}} q_{j+1,\varphi_{j+1}}^{(\ell_{j+1})}$, for some $j \geq 0$,
4. $q_{n-1}^{(\ell_{n-1})} a^{n_{n-1}} q_{n-1,\varphi_{n-1}}^{(\ell_{n-1})} q_f^{(\ell_n)} a^{n_n} q_f^{(\ell_n)}$ or $q_{n-1}^{(\ell_{n-1})} a^{n_{n-1}} q_{n-1,\varphi_{n-1}}^{(\ell_{n-1})} q_{f'}^{(\ell_n)} a^{n_n} q_{f'}^{(\ell_n)}$,
5. $q_{f'}^{(\ell_n)} a^{n_n} q_{f'}^{(\ell_n)} q_f^{(1)} a^{n_n} q_f^{(1)}$,
6. $q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)} q_1^{(\ell_1)} a^{n_1} q_{1,\varphi_1}^{(\ell_1)}$,
7. $q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)} q_f^{(\ell_1)} a^{n_1} q_f^{(\ell_1)}$ or $q_0^{(\ell_0)} a' q_{0,\varphi_0}^{(\ell_0)} q_{f'}^{(\ell_1)} a^{n_1} q_{f'}^{(\ell_1)}$.

We now describe how each such block can be accepted by a quasi real-time REV-DCA(1). A quasi real-time REV-DCA(1) accepting blocks of the first form basically increases the counter for every a (or a' if $n = 1$) from the first part and decreases the counter for every second a from the second part. This can be done reversibly, since in the backward computation every other a from the second part increases the counter while every a (or a' if $n = 1$) from the first part decreases the counter. We note that the counter is empty at the end of each forward computation. A quasi real-time REV-DCA(1) accepting blocks of the second form can similarly be constructed. The basic difference is that after reading $q_0^{(2)}$ in another component of the state set a counter modulo $1/k_0$ is started, if $k_0 < 1$. We recall that $k_0 \in C$ is the number associated with state q_0. While reading a's this counter is updated and finally compared with φ_0 when reading $q_{0,\varphi_0}^{(2)}$. If $k_0 \geq 1$, then no counter is started and it is only checked whether φ_0 is 0 when reading $q_{0,\varphi_0}^{(2)}$. This additional behavior can be realized reversibly.

For blocks of the third form we first note that we can reversibly check whether φ_j and φ_{j+1} are correctly computed with respect to a^{n_j} and $a^{n_{j+1}}$ by adapting the method described for blocks of the second form. Let (q_j, k_j, p, r) be the transition rule for state q_j. Now, a quasi real-time REV-DCA(1) increases its counter for every a from the first part. If $\varphi_j = 0$, we know that $q_{j+1} = p$ and $\ell_{j+1} = k_j > 1$ or n_j is divisible by $1/k_j$, if $k_j < 1$. If $k_j > 1$, then we decrease the counter for every k_j-th a from the second part. If $k_j < 1$, then we decrease the counter by $1/k_j$ within $1/k_j$ time steps on every a. This is realized by $1/k_j - 1$ stationary moves on every a. If $\varphi_j > 0$, we know that $q_{j+1} = r$ and $n_j = n_{j+1}$. Hence, we decrease the counter for every a from the second part. This can be done reversibly, since in the backward computation we know due to the information ℓ_{j+1} what to do on the counter. If $\ell_{j+1} = 1$, then the counter is increased for every a from the second part. If $\ell_{j+1} > 1$, then the counter is increased for every k_j-th a from the second part. If $\ell_{j+1} < 1$, then the counter is increased by $1/k_j$ within $1/k_j$ time steps for every a from the second part. Subsequently, the counter is decreased for every a from the first part. Altogether, a quasi real-time REV-DCA(1) accepting blocks of the third form can be constructed. We note that the counter is empty at the end of each forward computation. A quasi real-time REV-DCA(1) accepting blocks of the fourth form can similarly be constructed. The only difference is to replace $q_{j+1}^{(\ell_{j+1})}$ and $q_{j+1,\varphi_{j+1}}^{(\ell_{j+1})}$ by $q_f^{(\ell_n)}$ or $q_{f'}^{(\ell_n)}$, respectively. Finally, by using similar ideas we can also construct quasi real-time REV-DCA(1)s accepting blocks of the remaining forms.

Since every block can be accepted by a quasi real-time REV-DCA(1) and the counter is empty at the end of each computation, we can iterate these automata and obtain a quasi real-time REV-DCA(1) for $\mathrm{VALC}_1'(M)$ which can be sped-up to a real-time REV-DCA(1) owing to Theorem 3.2. □

Lemma 4.4. *Let M be a multiplying counter machine. Then a real-time* REV-DCA(2) *accepting the set* $\mathrm{VALC}'(M)$ *can effectively be constructed from M.*

Proof. We consider the real-time REV-DCA(1) M_1 and M_2 constructed in the proof of Lemma 4.3 that accept $\mathrm{VALC}_1'(M)$ and $\mathrm{VALC}_2'(M)$, respectively. We note that in all accepting computations in M_1 as well as in M_2 the input has been completely read. Hence, we can apply the well known Cartesian product technique for intersection and construct a real-time REV-DCA(2) M' that simulates M_1 in one component of the state set and uses one counter and and simulates M_2 in a second component of the state set and uses the other counter. The accepting states of M' are defined as $F_1 \times F_2$, where F_1 and F_2 are the accepting states in M_1 and M_2, respectively. Hence, M' accepts $\mathrm{VALC}_1'(M) \cap \mathrm{VALC}_2'(M) = \mathrm{VALC}'(M)$. □

Now, we have all preparatory results to show the following non-semidecidability results.

Theorem 4.5. *Let M and M' be two real-time* REV-DCA(k) *with $k \geq 2$. Then the following questions are not semidecidable.*

1. *Is $L(M) = \emptyset$?*
2. *Is $L(M)$ finite/infinite?*
3. *Is $L(M) \subseteq L(M')$?*
4. *Is $L(M) = L(M')$?*
5. *Is $L(M)$ regular/context free?*

Proof. Let T be some Turing machine accepting a recursively enumerable set over $\{1,2\}^*$ and M_0 its corresponding multiplying counter machine according to Theorem 4.1. Then, the language $L(T)$ is empty if and only if M_0 accepts the empty set. Moreover, M_0 accepts the empty set if and only if $\mathrm{VALC}'(M_0) = \emptyset$ and T accepts a finite set if and only if M_0 accepts a finite set if and only if $\mathrm{VALC}'(M_0)$ is finite.

Now, let M_1 be a real-time REV-DCA(2) accepting $\mathrm{VALC}'(M_0)$ according to the construction in Lemma 4.4. Hence, $L(M_1)$ is empty, finite, or infinite if and only if the Turing machine T accepts an empty, finite, or infinite set. Since the latter questions are not semidecidable for Turing machines (see, *e.g.*, [23]), they are not semidecidable for real-time REV-DCA(2) as well.

It is easy to construct a real-time REV-DCA(2) accepting the empty set. If the questions of inclusion and equivalence would be semidecidable, the question of emptiness would be semidecidable as well which is a contradiction. Hence, both questions are not semidecidable.

It is described in [23] how to define the set of valid computations $\mathrm{VALC}(T)$ of a Turing machine T. In addition, it is shown there with the help of the pumping lemma that $\mathrm{VALC}(T)$ is not a context-free language if T accepts an infinite language. It can be shown with a similar approach that $\mathrm{VALC}'(M_0)$ is not a context-free language if M_0 accepts an infinite language. On the other hand, if M_0 accepts a finite language, then $\mathrm{VALC}'(M_0)$ is finite and hence in particular a regular and a context-free language. Altogether, we have that $\mathrm{VALC}'(M_0)$ is finite if and only if M_0 is regular or context free. If the regularity or context-freeness of a real-time REV-DCA(2) would be semidecidable, we would therefore obtain that the finiteness problem for a real-time REV-DCA(2) is semidecidable as well which is a contradiction and shows the remaining claim of the theorem. □

In case of counter machines with one counter, real-time REV-DCA(1) can be considered as deterministic pushdown automata. Since for the latter the questions of emptiness, universality, finiteness, infiniteness, equivalence, and regularity are decidable [23, 25, 26], we obtain that all these questions are decidable for real-time REV-DCA(1) as well. The question of inclusion is known to be undecidable for deterministic pushdown automata. Next, we will show that the problem remains undecidable for real-time REV-DCA(1) as well. To show this result we first need a technical lemma and the result that real-time REV-DCA(1) are closed under complementation.

Lemma 4.6. *Let M be a real-time* REV-DCA(1).

1. *Every computation that contains at least three stationary moves is non-accepting.*
2. *Every accepting computation that contains two stationary moves, necessarily performs a stationary move as last move.*

Proof. Let $w = \triangleright a_1 a_2 \cdots a_n \triangleleft$ be an input on which M performs an accepting computation with at least three stationary moves. For technical reasons, we denote $\triangleright$ by a_0. Since M accepts in real time, M can perform at most $n+2$ moves before accepting. Since M performs at least three stationary moves, it may perform at most $n-1$ right moves. Hence, the last right move must be performed on a_i with $0 \leq i < n-1$. Since M performs $i+1$ right moves with $0 \leq i < n-1$ and three stationary moves, M halts accepting after $i+4$ steps having performed the last right move on some symbol a_i with $0 \leq i < n-1$, or M halts accepting after three stationary moves on $\triangleright$. In the latter case, M halts accepting on input $\triangleright a_1 \triangleleft$ after three stationary moves without leaving $\triangleright$.

Hence, M halts accepting as well after three steps on input $\rhd\lhd$. However, a real-time computation on input $\rhd\lhd$ may perform at most two steps. This is a contradiction.

In the first case, we conclude that on input $w' = \rhd a_1 a_2 \cdots a_i a_{i+1} \lhd$ M halts accepting as well after $i+4$ steps. However, a real-time computation on input w' may perform at most $i+3$ steps. This is again a contradiction and concludes the proof of the first part of the lemma.

Now, we consider an accepting computation of M on input w with two stationary moves. Since M accepts in real time and M performs two stationary moves, it may perform at most n right moves. In addition, we assume that the last move before halting has been a right move. Hence, the last right move must be performed on a_i with $0 \leq i < n$. Since M performs $i+1$ right moves with $0 \leq i < n$ and two stationary moves, M halts accepting after $i+3$ steps having performed the last right move on some symbol a_i with $0 \leq i < n$. But then M cannot process the rest of the input. In particular, on input $w'' = \rhd a_1 a_2 \cdots a_i \lhd$ the automaton M halts accepting as well after $i+3$ steps. However, a real-time computation on input w'' may perform at most $i+2$ steps. This contradiction concludes the proof of the second part of the lemma. □

Lemma 4.7. *Let M be a real-time* REV-DCA(1). *Then, a real-time* REV-DCA(1) *accepting the complement $\overline{L(M)}$ can effectively be constructed.*

Proof. The basic idea is to interchange accepting and non-accepting states. To this end, it is necessary that any computation of M halts within real time, *i.e.*, after at most $n+2$ time steps on inputs of length n. We will construct a modified real-time REV-DCA(1) M' based on M. Let Q be the state set of M, let Q' and Q'' be each a copy of Q, let the state set of M' be $Q \cup Q' \cup Q''$, and let M' initially have the same transitions as M.

We first consider computations of M in which no stationary moves are made. Clearly, such computations halt within real time.

Second, we consider computations in which exactly one stationary move is made. To halt within real time it is in this case sufficient to prevent M' from reading the right endmarker when the stationary move has been performed. If there is a stationary transition $\delta(p,x,c) = (q,0,u)$ with $p,q \in Q$, $x \in \Sigma \cup \{\rhd, \lhd\}$, $c \in \{+,\bot\}$, and $u \in \{+1,0,-1\}$, then we modify this transition in M' to $\delta'(p,x,c) = (q',0,u)$. Furthermore, for non-stationary moves the automaton behaves on states from Q' the same way as on states from Q. Formally, if M contains a transition $\delta(p,x,c) = (q,1,u)$ with $p,q \in Q$, $x \in \Sigma \cup \{\rhd, \lhd\}$, $c \in \{+,\bot\}$, and $u \in \{+1,0,-1\}$, then we add in M' a transition $\delta'(p',x,c) = (q',1,u)$. Finally, we remove in M' all transitions $\delta'(p', \lhd, c) = (q',1,u)$ for some $p', q' \in Q'$, $c \in \{+,\bot\}$, and $u \in \{+1,0,-1\}$. Altogether, all computations with exactly one stationary move are halting in M' within real time.

Third, we consider computations in which at least two stationary moves are made. Due to Lemma 4.6 we know that such computations are either non-accepting or halt accepting after having performed the second stationary move. Owing to the changes made for the second case, a state in Q' indicates that one stationary move has already been performed. Hence, we complement the modifications already made with new modifications in order to send a computation with a second stationary move into a halting, accepting state, if the computation in M is accepting, and into a halting, non-accepting state otherwise. The latter modification can be done, since we know due to Lemma 4.6 that any second stationary move not leading to an accepting, halting configuration belongs to a non-accepting computation. Formally, let $\delta(p,x,c) = (q,0,u)$ with $p,q \in Q$, $x \in \Sigma \cup \{\rhd\}$, $c \in \{+,\bot\}$, and $u \in \{+1,0,-1\}$ be a transition in M. If q is accepting, we add in M' a transition $\delta'(p',x,c) = (q'',0,u)$. Depending on the counter height, it may happen that M' enters an accepting, non-halting configuration. In this case, we have to halt non-accepting and hence add in M' a transition $\delta'(q'',y,c') = ((q'',y,c'),0,0)$, where $(q'',y,c') \in Q'' \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$ is a new, halting and non-accepting state and M contains a transition for the triple $(q,y,c') \in Q \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$. If q is not accepting, we add in M' a transition $\delta'(p',x,c) = ((p',x,c),0,0)$, where $(p',x,c) \in Q' \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$ is a new, halting and non-accepting state. We note that we add no transitions, if the second stationary move in M is made on the right endmarker. This can be done without harm, since a second stationary move on $\lhd$ drives M beyond real time. Altogether, we have achieved that a computation in M' halts after at most two stationary moves where at most one is performed on the right

endmarker. To show that the modified automaton halts in real time, we distinguish the following cases:

1. Both stationary moves in M are made on symbols different from $\lhd$. In this case at most n symbols of the input have been read and the computation in M' stops after at most $n+2$ steps.
2. Exactly one stationary move in M is made on a symbol different from $\lhd$. In this case, $n+1$ symbols of the input have been read and one stationary move has been made, when the reading head sees $\lhd$ for the first time. Since there are no stationary transitions from Q' for the right endmarker, the computation in M' stops after $n+2$ steps.
3. Both stationary moves in M are made on $\lhd$. In this case, $n+1$ symbols of the input have been read, when the reading head sees $\lhd$ for the first time. The first stationary move on the right endmarker is realized with the modifications in the above second case and a state in Q' is entered. Since there are no stationary transitions from Q' for the right endmarker, the computation in M' again stops after $n+2$ steps.

Finally, we have to show that the modifications made preserve reversibility. The added transitions from a state in Q' to a state in $Q' \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$ are reversible, since for every $(p',x,c) \in Q' \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$ there is at most one transition ending in state (p',x,c) and the counter is not touched. The same reasoning holds for the added transitions from a state in Q'' to a state in $Q'' \times (\Sigma \cup \{\rhd\}) \times \{+,\bot\}$. The added transitions from a state in Q' to a state in Q' are reversible, since the original transitions from a state in Q to a state in Q are reversible. The modified transitions from a state in Q to a state in Q' are reversible, since a stationary transition $\delta(p,x,c) = (q,0,u)$ with $p,q \in Q$, $x \in \Sigma \cup \{\rhd,\lhd\}$, $c \in \{+,\bot\}$, and $u \in \{+1,0,-1\}$ in M has a corresponding backward stationary transition $\delta^{\leftarrow}(q,x,c') = (p,0,-u)$ with $c' \in \{+,\bot\}$ in M. Hence, if M' is in state $q' \in Q'$ and M would perform a stationary backward move, we can enter the state and change the counter accordingly. A similar reasoning can be made for the added transitions from a state in Q' to a state in Q''. Finally, the removal of transitions does not affect the reversibility.

Altogether, the modifications result in a real-time REV-DCA(1) such that any computation halts within real time. Hence, a real-time REV-DCA(1) accepting the complement can be constructed by just interchanging accepting and non-accepting states. □

Now, we have all prerequisites and can show the non-semidecidability of the inclusion problem for real-time REV-DCA(1).

Theorem 4.8. *Let M, M' be two real-time* REV-DCA(1)*. Then the inclusion $L(M) \subseteq L(M')$ is not semidecidable.*

Proof. Let T be some Turing machine accepting a recursively enumerable set over $\{1,2\}^*$ and M_0 its corresponding multiplying counter machine according to Theorem 4.1. Then language $L(T)$ is empty if and only if M_0 accepts the empty set. Moreover, M_0 accepts the empty set if and only if $\mathrm{VALC}'(M_0) = \emptyset$.

Now, let M and M' be the two real-time REV-DCA(1) accepting $\mathrm{VALC}'_1(M_0)$ and $\mathrm{VALC}'_2(M_0)$, respectively, according to the construction in Lemma 4.4. Since the class of REV-DCA(1) is closed under complementation due to Lemma 4.7, a REV-DCA(1) M'' accepting $\overline{L(M')}$ can be constructed.

In contrast to the assertion, we assume that the inclusion problem is semidecidable. Then the inclusion $L(M) \subseteq L(M'') = \overline{L(M')}$ is semidecidable. This is equivalent to semidecide $L(M) \cap L(M') = \emptyset$ which implies that the emptiness of $\mathrm{VALC}(M_0)$, and hence of $L(T)$, is semidecidable. This is a contradiction. □

It is shown in [14] by a reduction from Post's Correspondence Problem that the inclusion problem for reversible deterministic pushdown automata is undecidable. Since real-time REV-DCA(1) can be considered as reversible DPDAs, we yield the following improvement, namely, that the inclusion problem for reversible DPDAs is even not semidecidable.

Corollary 4.9. *Let M and M' be two reversible* DPDA*s. Then the inclusion $L(M) \subseteq L(M')$ is not semidecidable.*

5. Conclusions

In this paper, we investigated the feature of reversibility in one-way multi-counter automata. The basic results are the separation of the classes of languages accepted by irreversible and reversible k-counter automata with polynomial as well as superpolynomial time bounds. In addition, tight infinite counter hierarchies are obtained for automata working in polynomial time as well as working in exponential time. These results imply also infinite counter hierarchies for not necessarily reversible counter automata and, therefore, extend known results. Finally, the decidability status for the usually studied decidability questions could fully be settled. All these questions are not even semidecidable for reversible real-time automata with at least two counters. In case of reversible real-time one-counter automata, we have the decidability of the questions of emptiness, finiteness, infiniteness, equivalence, and regularity, but could show the non-semidecidability of the inclusion problem.

Some topics for future research are as follows. First of all, it would be of interest to determine the closure properties of the language classes discussed in this paper. Second, Ibarra introduced and studied in [27] multi-counter automata which are restricted in such a way that the number of changes between increasing and decreasing a counter is bounded by some constant. It would clearly be interesting to add the feature of reversibility to these restricted variants. Finally, we have investigated so far only one-way multi-counter automata. Thus, it is an immediate task to add the feature of reversibility also to multi-counter automata with two-way motion of the head. An interesting starting point could be the study of *sweeping* multi-counter automata which means that the direction of the head can only change at both ends of the input.

References

[1] C.H. Bennett, Logical reversibility of computation. *IBM J. Res. Dev.* **17** (1973) 525–532.
[2] Y. Lecerf, Logique mathématique: Machines de Turing réversible. *C. R. Séances Acad. Sci.* **257** (1963) 2597–2600.
[3] K. Morita, Reversible simulation of one-dimensional irreversible cellular automata. *Theor. Comput. Sci.* **148** (1995) 157–163.
[4] D. Angluin, Inference of reversible languages. *J. ACM* **29** (1982) 741–765.
[5] A. Kondacs and J. Watrous, On the power of quantum finite state automata. in *FOCS 1997*. IEEE Computer Society (1997) 66–75.
[6] M. Holzer, S. Jakobi and M. Kutrib, Minimal reversible deterministic finite automata. *Int. J. Found. Comput. Sci.* **29** (2018) 251–270.
[7] G.J. Lavado, G. Pighizzini and L. Prigioniero, Minimal and reduced reversible automata. *J. Autom. Lang. Comb.* **22** (2017) 145–168.
[8] G.J. Lavado and L. Prigioniero, Concise representations of reversible automata. *Int. J. Found. Comput. Sci.* **30** (2019) 1157–1175.
[9] A. Bertoni and M. Carpentieri, Regular languages accepted by quantum automata. *Inf. Comput.* **165** (2001) 174–182.
[10] M. Hirvensalo, Quantum automata with open time evolution. *Int. J. Nat. Comput. Res.* **1** (2010) 70–85.
[11] A.C.C. Say and A. Yakaryılmaz, Quantum finite automata: A modern introduction, edited by C.S. Calude, R. Freivalds and K. Iwama. Computing with New Resources. Vol. 8808 of *LNCS*. Springer (2014) 208–222.
[12] Ö. Yilmaz, F. Kiyak, M. Üngör and A.C.C. Say, Energy complexity of regular language recognition. edited by P. Caron and L. Mignot. Implementation and Application of Automata (CIAA 2022). Vol. 13266 of *LNCS*. Springer (2022) 200–211.
[13] C. Mereghetti and B. Palano, Guest column: quantum finite automata: from theory to practice. *SIGACT News* **52** (2021) 38–59.
[14] M. Kutrib and A. Malcher, Reversible pushdown automata. *J. Comput. Syst. Sci.* **78** (2012) 1814–1827.
[15] M.L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Ann. Math.* **74** (1961) 437–455.
[16] S.A. Greibach, Remarks on the complexity of nondeterministic counter languages. *Theor. Comput. Sci.* **1** (1976) 269–288.
[17] H. Petersen, Simulations by time-bounded counter machines. *Int. J. Found. Comput. Sci.* **22** (2011) 395–409.
[18] L. Breveglieri, A. Cherubini, C. Citrini and S. Crespi-Reghizzi, Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.* **7** (1996) 253–292.
[19] K. Morita, Universality of a reversible two-counter machine. *Theor. Comput. Sci.* **168** (1996) 303–320.
[20] P.C. Fischer, A.R. Meyer and A.L. Rosenberg, Counter machines and counter languages. *Math. Syst. Theory* **2** (1968) 265–283.
[21] M. Li and P.M.B. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications. Springer (1993).
[22] R. Laing, Realization and complexity of commutative events. Technical Report 03105-48-T, University of Michigan (1967).
[23] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979).
[24] J. Hartmanis and J.E. Hopcroft, What makes some language theory problems undecidable. *J. Comput. Syst. Sci.* **4** (1970) 368–376.

[25] G. Sénizergues, $L(A) = L(B)$? A simplified decidability proof. *Theor. Comput. Sci.* **281** (2002) 555–608.
[26] R.E. Stearns, A regularity test for pushdown machines. *Inform. Control* **11** (1967) 323–340.
[27] O. Ibarra, Reversal-bounded multicounter machines and their decision problems. *J. ACM* **25** (1978) 116–133.