

# Lower bounds for processing data with few random accesses to external memory

Martin Grohe<sup>1</sup>      André Hernich<sup>2</sup>      Nicole Schweikardt<sup>2</sup>

<sup>1</sup> Institut für Informatik, Humboldt-Universität zu Berlin, Germany  
grohe@informatik.hu-berlin.de

<sup>2</sup> Institut für Informatik, Goethe-Universität Frankfurt am Main, Germany  
{hernich|schweika}@informatik.uni-frankfurt.de

## Abstract

We consider a scenario where we want to query a large dataset that is stored in external memory and does not fit into main memory. The most constrained resources in such a situation are the size of the main memory and the number of random accesses to external memory. We note that sequentially streaming data from external memory through main memory is much less prohibitive.

We propose an abstract model of this scenario in which we restrict the size of the main memory and the number of random accesses to external memory, but admit arbitrary sequential access. A distinguishing feature of our model is that it allows the usage of unlimited external memory for storing intermediate results, such as several hard disks that can be accessed in parallel.

In this model, we prove lower bounds for the problem of sorting a sequence of strings (or numbers), the problem of deciding whether two given sets of strings are equal, and two closely related decision problems. Intuitively, our results say that there is no algorithm for the problems that uses internal memory space bounded by  $N^{1-\varepsilon}$  and at most  $o(\log N)$  random accesses to external memory, but unlimited “streaming access”, both for writing to and reading from external memory. (Here  $N$  denotes the size of the input and  $\varepsilon$  is an arbitrary constant greater than 0.) We even permit randomized algorithms with one-sided bounded error. We also consider the problem of evaluating database queries and prove similar lower bounds for evaluating relational algebra queries against relational databases and XQuery and XPath queries against XML-databases.

## 1 Introduction

The massive datasets that have to be processed in many applications are often far too large to fit completely into the computer’s (internal) main memory and thus have to reside in external memory (such as disks). When querying such data, the most constrained resources are the size of the main memory and the number of random accesses to external memory. It is well-known that access to external memory is by orders of magnitude slower than access to main memory. However, there is an important distinction to be made between a random access to data in a particular external memory location, which involves moving the head to that location, and sequentially streaming data off the disks. Modern software and database technology uses clever heuristics to minimize the number of accesses to external memory and to prefer streaming over random accesses to external memory.

There has been a wealth of research on the design of so-called *external memory algorithms* (cf., e.g. [20, 23, 17]). In recent years, in the context of data stream applications some efforts have been made to study the limitations of answering queries in a setting where only one or few sequential passes over the data are admitted [19, 2, 15, 4, 20, 5, 1, 13]. They result in strong lower bounds for a number of natural algorithmic problems and for answering database queries. The model underlying all these lower bounds allows few sequential passes over the data (or just one sequential pass in the important special case of data streams) and apart from that no access to external memory. The query processing

takes place entirely in main memory, which of course is of limited size. The model does not allow for intermediate results to be stored in *auxiliary external memory*. However, storing intermediate results in external memory can be a very powerful mechanism. For example, while the data are sequentially read in a first pass, an algorithm may annotate the data and write the annotation on a second disk. Then in a second pass, the algorithm may use the annotation to compute the answer to the query. Or an algorithm may copy the first half of the data onto a second disk and then read the copied first half of the data in parallel with the second half, maybe to merge the two halves or compute a join.

## 1.1 The model

We prove lower bounds for various algorithmic problems, including sorting and query answering, in a streaming model with auxiliary external memory. Our model is a natural extension of a model introduced in [13] to the setting with auxiliary external memory devices. Recall that the two most significant cost measures in this setting are the number of random accesses to external memory and the size of the internal memory. The model is based on a standard multi-tape Turing machine. Some of the tapes of the machine, among them the input tape, represent the external memory. They are unrestricted in size, but access to these tapes is restricted by allowing only a certain number  $r(N)$  (where  $N$  denotes the input size) of reversals of the head directions. This may be seen as a way of (a) restricting the number of sequential scans and (b) restricting random access to these tapes, because each random access can be simulated by moving the head to the desired position on a tape, which involves at most two head reversals. The remaining tapes of the Turing machine represent the internal memory. Access to these internal memory tapes (i.e., the number of head reversals) is unlimited, but their size is bounded by a parameter  $s(N)$ . We let  $\text{ST}(r(N), s(N), O(1))$  denote the class of all problems that can be solved on such an  $(r(N), s(N), O(1))$ -bounded Turing machine, i.e., a Turing machine with an arbitrary number of external memory tapes (the third parameter denotes the number of external memory tapes, which is here  $O(1)$ ) which, on inputs of size  $N$ , performs less than  $r(N)$  head reversals on the external memory tapes, and uses at most space  $s(N)$  on the internal memory tapes. We also consider a randomized version of our ST-classes. To this end, we introduce the complexity class  $\text{RST}(r(N), s(N), O(1))$ , which consists of all decision problems that can be solved by an  $(r(N), s(N), O(1))$ -bounded randomized Turing machine with one-sided bounded error, where no false positive answers are allowed and the probability of false negatives is at most  $1/2$ . To be able to deal with problems where an output (other than just a yes/no answer) has to be generated, we introduce a class  $\text{LasVegas-RST}(r(N), s(N), O(1))$  to denote the class of all functions  $f$  for which there exists an  $(r(N), s(N), O(1))$ -bounded randomized Turing machine that with probability at least  $1/2$  computes the correct result, but may also produce no result at all (and terminate with a statement like “I don’t know”).

## 1.2 Results

We study the problem SORT of sorting a sequence of strings (or numbers), the problem SET-EQUALITY of deciding whether two given sets of strings are equal, and two closely related decision problems, MULTI-SET-EQUALITY and CHECK-SORT. Furthermore, we study the evaluation of database queries both in classical relational databases and in the XML-context.

In the following,  $N$  always denotes the size of the input. We prove that for every  $\varepsilon > 0$

$$\text{SET-EQUALITY} \notin \text{RST}(o(\log N), N^{1-\varepsilon}, O(1)).$$

Intuitively, this means that there is no algorithm for the SET-EQUALITY problem that uses internal memory space bounded by  $N^{1-\varepsilon}$  and at most  $o(\log N)$  random accesses to external memory, but unlimited “streaming access”, both for writing to and reading from external memory. Actually, we prove a slightly stronger lower bound that shows a trade-off between head reversals and space; the precise statement can be found in Theorem 3.2. Similarly, we prove that

$$\text{SORT} \notin \text{LasVegas-RST}(o(\log N), N^{1-\varepsilon}, O(1)).$$

We also obtain matching upper bounds by putting both SET-EQUALITY and SORT into  $ST(O(\log N), O(1), 2)$ . We obtain the same results as for SET-EQUALITY for MULTI-SET-EQUALITY and CHECK-SORT.

As a corollary of the lower bound for SET-EQUALITY, we obtain similar lower bounds for answering relational algebra queries against relational databases and XQuery and XPath queries against XML-databases. For relational algebra queries, we observe that there is a matching upper bound of  $ST(O(\log N), O(1), O(1))$ .

Using standard fingerprinting techniques, we show that MULTI-SET-EQUALITY belongs to the class  $\text{co-RST}(2, O(\log N), 1)$ . This separates  $\text{RST}(r, s, O(1))$  from  $\text{co-RST}(r, s, O(1))$  and thus both from the deterministic  $ST(r, s, O(1))$  for a wide range of parameters  $r, s$ . We also separate the randomized classes from the corresponding nondeterministic classes.

### 1.3 Discussion and related work

Strong lower bounds for a number of problems are known in the context of data streams and for models which permit a small number of sequential scans of the input data, but no auxiliary external memory [19, 2, 15, 4, 20, 5, 6, 1, 13]. All these lower bounds are obtained by communication complexity. Note that in the presence of at least two external memory tapes, communication between remote parts of memory is possible by simply copying data from one tape to another and then re-reading both tapes in parallel. These communication abilities of our model spoil any attempt to prove lower bounds via communication complexity.

To prove our lower bounds, we introduce an auxiliary computation model, which we call *list machine*. We prove that  $(r, s, t)$ -bounded Turing machines can be simulated by list machines that are restricted in a similar way. The list machines admit a clearer view of the flow of information during a computation, and this allows us to prove lower bounds for list machines by direct combinatorial arguments.

Obviously, our model is related to the *bounded reversal Turing machines*, which have been studied in classical complexity theory (see, e.g., [24, 9]). However, in bounded reversal Turing machines, the number of head reversals is limited on *all* tapes, whereas in our model there is no such restriction on the internal memory tapes. This makes our model considerably stronger, considering that in our lower bound results we allow internal memory size that is close to the input size. Furthermore, to the best of our knowledge, all lower bound proofs previously known for reversal complexity classes on multi-tape Turing machines go back to the space hierarchy theorem (cf., e.g., [21, 9]) and thus rely on diagonalization arguments, and apply only to classes with  $\omega(\log N)$  head reversals. In particular, these lower bounds do not include the checksort problem and the (multi)set equality problem, as these problems can be solved with  $O(\log N)$  head reversals.

In the classical *parallel disk model* for external memory algorithms (see, e.g., [23, 17, 20]), the cost measure is simply the number of bits read from external memory divided by the page size. Several refinements of this model have been proposed to include a distinction between *random access* and *sequential scans* of the external memory, among them Arge and Bro Miltersen's *external memory Turing machines* [3]. We note that their notion of external memory Turing machines significantly differs from ours, as their machines only have a single external memory tape and process inputs that consist of a *constant* number  $m$  of input strings. Strong lower bound results (in particular, for different versions of the *sorting problem*) are known for the parallel disk model (see [23] for an overview) as well as for Arge and Bro Miltersen's external memory Turing machines [3]. However, all these lower bound proofs heavily rely on the assumption that the input data items (e.g., the strings that are to be sorted) are *indivisible* and that at any point in time, the external memory consists, in some sense, of a permutation of the input items. We emphasize that the present paper's lower bound proofs do not rely on such an indivisibility assumption.

The second and third author have obtained further results on the structural complexity of the ST-classes in [16]. In particular, these results show that as soon as we admit  $\Omega(\log N)$  head reversals or nondeterminism, the classes get very large. For example, it is observed in [16] that  $\text{LOGSPACE} \subseteq \text{ST}(O(\log N), O(1), O(1))$  and that  $\text{NP} = \text{NST}(O(\log N), O(1), O(1)) = \text{NST}(O(1), O(\log N), O(1))$ .

The present article combines the results of the two conference papers [14, 11]; let us remark that the lower bounds are considerably strengthened here. [12] is an introductory survey to the results of the present article and [13]; a more detailed survey can be found in [22]. Recently, lower bound results

for  $(r(N), s(N), O(1))$ -bounded randomized Turing machines with *2-sided* bounded error have been obtained in [8, 7].

## 1.4 Organization

After introducing the  $ST(\dots)$  classes in Section 2, we formally state our main results in Section 3 and give some easy proofs. The subsequent sections are devoted to proving the lower bound results. In Section 4, 5, and 6 we introduce list machines, show that Turing machines can be simulated by list machines, and prove that randomized list machines can neither solve the *(multi)set equality problem* nor the *checksort problem*. Afterwards, in Section 7 we transfer these results from list machines to Turing machines. Finally, in Section 8 we present a lemma that reduces the problem of proving lower bounds for  $(r, s, t)$ -bounded Turing machines to a purely *combinatorial* problem.

## 2 Complexity Classes

We write  $\mathbb{N}$  to denote the set of natural numbers (that is, nonnegative integers).

As our basic model of computation, we use standard multi-tape nondeterministic Turing machines (NTMs, for short); cf., e.g., [21]. The Turing machines we consider will have  $t + u$  tapes. We call the first  $t$  tapes *external memory tapes* (and think of them as representing  $t$  disks). We call the other  $u$  tapes *internal memory tapes*. The first tape is always viewed as the input tape. Our precise notation is as follows:

### Definition 2.1.

Let  $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$  be a nondeterministic Turing machine (NTM, for short) with  $t + u$  tapes, where  $Q$  is the state space,  $\Sigma$  the alphabet,  $q_0 \in Q$  the start state,  $F \subseteq Q$  the set of final states,  $F_{acc} \subseteq F$  the set of *accepting* states, and

$$\Delta \subseteq (Q \setminus F) \times \Sigma^{t+u} \times Q \times \Sigma^{t+u} \times \{L, N, R\}^{t+u}$$

the transition relation. Here  $L, N, R$  are special symbols indicating the head movements.

We assume that all tapes are one-sided infinite and have cells numbered  $1, 2, 3$ , etc, and that  $\square \in \Sigma$  is the “blank” symbol which, at the beginning of the TM’s computation, is the inscription of all empty tape cells.

A *configuration* of  $T$  is a tuple

$$(q, p_1, \dots, p_{t+u}, w_1, \dots, w_{t+u}) \in Q \times \mathbb{N}^{t+u} \times (\Sigma^*)^{t+u},$$

where  $q$  is the current state,  $p_1, \dots, p_{t+u}$  are the positions of the heads on the tapes, and  $w_1, \dots, w_{t+u}$  are the contents of the tapes. For a configuration  $\gamma$  we write  $\text{Next}_T(\gamma)$  for the set of all configurations  $\gamma'$  that can be reached from  $\gamma$  in a single computation step.

A configuration is called *final* (resp., *accepting*, *rejecting*) if its current state  $q$  is final (resp., accepting, final and not accepting), that is,  $q \in F$  (resp.,  $q \in F_{acc}$ ,  $q \in F \setminus F_{acc}$ ). Note that a final configuration does not have a successor configuration.

A *run* of  $T$  is a sequence  $\rho = (\rho_j)_{j \in J}$  of configurations  $\rho_j$  satisfying the obvious requirements. We are only interested in finite runs here, where the index set  $J$  is  $\{1, \dots, \ell\}$  for an  $\ell \in \mathbb{N}$ , and where  $\rho_\ell$  is final.

When considering decision problems, a run  $\rho$  is called *accepting* (resp., *rejecting*) if its final configuration is accepting (resp., rejecting). When considering, instead, Turing machines that produce an output, we say that a run  $\rho$  *outputs the word*  $w'$  if  $\rho$  ends in an accepting state and  $w'$  is the inscription of the last (i.e.,  $t$ -th) external memory tape. If  $\rho$  ends in a rejecting state, we say that  $\rho$  *outputs “I don’t know”*.

Without loss of generality we assume that our Turing machines are *normalized* in such a way that in each step at most one of its heads moves to the left or to the right. ⊣

Let  $T$  be an NTM with  $t + u$  tapes, and let  $\rho = (\rho_1, \dots, \rho_\ell)$  be a run of  $T$ . For every  $1 \leq i \leq t + u$  and  $1 \leq j \leq \ell$ , the *direction* of the  $i$ th head in step  $j$  is defined inductively by letting  $d_{i1} = 1$  and, for  $j \geq 2$ ,

$$d_{ij} = \begin{cases} 1 & \text{if } p_{i(j-1)} < p_{ij} \\ -1 & \text{if } p_{ij} < p_{i(j-1)} \\ d_{i(j-1)} & \text{otherwise.} \end{cases}$$

Here  $p_{ij}$  denotes the position of the  $i$ th head in step  $j$ . We say that the  $i$ th head *changes its direction* in step  $j \geq 2$  if  $d_{ij} \neq d_{i(j-1)}$ , and we let  $\text{rev}(\rho, i)$  denote the number of times the  $i$ th head changes its direction in the run  $\rho$ . Furthermore, we let  $\text{space}(\rho, i)$  be the number of cells of tape  $i$  that are used by  $\rho$ .

**Definition 2.2 (( $r, s, t$ )-bounded TM).** Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ . A (nondeterministic) Turing machine  $T$  is ( $r, s, t$ )-*bounded*, if every run  $\rho$  of  $T$  on an input of length  $N$  (for arbitrary  $N \in \mathbb{N}$ ) satisfies the following conditions:

- (1)  $\rho$  is finite,
- (2)  $1 + \sum_{i=1}^t \text{rev}(\rho, i) \leq r(N)$ , and
- (3)  $\sum_{i=t+1}^{t+u} \text{space}(\rho, i) \leq s(N)$ , where  $t + u$  is the total number of tapes of  $T$ . –

It is convenient for technical reasons to add 1 to the number  $\sum_{i=1}^t \text{rev}(\rho, i)$  of changes of the head direction in item (2) of the definition. As defined here,  $r(N)$  thus bounds the number of sequential scans of the external memory tapes rather than the number of changes of head directions.

Throughout this paper, we adopt the following convention: Whenever the letters  $r$  and  $s$  denote functions from  $\mathbb{N}$  to  $\mathbb{N}$ , these functions are *monotone*, i.e., we have  $r(x) \leq r(y)$  and  $s(x) \leq s(y)$  for all  $x, y \in \mathbb{N}$  with  $x \leq y$ .

**Definition 2.3 (The classes  $\text{ST}(\dots)$  and  $\text{NST}(\dots)$ ).**

Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ . Then  $\text{ST}(r, s, t)$  (resp.,  $\text{NST}(r, s, t)$ ) is the class of all problems that can be decided by a ( $r, s, t$ )-bounded (resp., nondeterministic) Turing machine. –

As it is common in complexity theory, we usually view  $\text{ST}(r, s, t)$  and  $\text{NST}(r, s, t)$  as classes of decision problems. However, we may still say that a functional problem  $f : \Sigma^* \rightarrow \Sigma^*$  belongs to  $\text{ST}(r, s, t)$ , meaning that there is a deterministic ( $r, s, t$ )-bounded Turing machine that computes  $f$ . We never put functional problems into the nondeterministic  $\text{NST}(r, s, t)$  classes, though, and structural results such as  $\text{ST}(r, s, t) \subseteq \text{NST}(r, s, t)$  always refer to decision problems.

Note that we put no restriction on the running time or the space used on the first  $t$  tapes of an ( $r, s, t$ )-bounded Turing machine. The following lemma shows that these parameters cannot get too large.

**Lemma 2.4.** *Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ , and let  $T$  be an ( $r, s, t$ )-bounded NTM. Then for every run  $\rho = (\rho_1, \dots, \rho_\ell)$  of  $T$  on an input of size  $N \geq 1$  we have  $\ell \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$  and thus  $\sum_{i=1}^t \text{space}(\rho, i) \leq N \cdot 2^{O(r(N) \cdot (t+s(N)))}$ . –*

*Proof:* Suppose that  $T = (Q, \Sigma, \delta, q_0, F)$  and that  $T$  has  $u$  internal memory tapes and thus  $t + u$  tapes in total. Let  $v \in \Sigma^*$  with  $|v| = N$  be the input of the run  $\rho$ .

Let  $\hat{Q}$  be the set of potential configurations of the tapes  $t+1, \dots, t+u$ , together with the current state of  $T$ , that is,

$$\hat{Q} = \left\{ (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}) : q \in Q, \right. \\ \left. p_{t+i} \in \{1, \dots, s(N)\}, w_{t+i} \in \Sigma^{\leq s(N)} \text{ for all } i \in \{1, \dots, u\} \right\}.$$

Note that since  $T$  is ( $r, s, t$ )-bounded, the tapes  $t+1, \dots, t+u$  always have length at most  $s(N)$ .

We have

$$|\hat{Q}| \leq |Q| \cdot s(N)^u \cdot (|\Sigma| + 1)^{s(N)} = 2^{O(s(N))}.$$

Note that  $T$  can perform at most  $|\Sigma|^t \cdot |\hat{Q}|$  steps without moving any of the heads on the first  $t$  tapes (otherwise, there would be “loop” in the run, which can be used to construct an infinite run of  $T$ , contradicting the assumption that every run is finite, cf. Definition 2.2). With the same reasoning, one even obtains the following statement: For each  $\tau$ , let  $k_\tau$  denote the current length of the string on tape  $\tau$  (i.e.,  $k_\tau$  is

the maximum position on tape  $\tau$  that has already been visited). Then  $T$  can make at most  $|\Sigma^t| \cdot |\hat{Q}|$  steps without moving any of the heads of a tape  $\tau \in \{1, \dots, t\}$  whose head is on a position  $\leq k_\tau$ .

It follows that, without changing the *direction* of a head on any of the tapes  $1, \dots, t$ ,  $T$  can perform at most

$$(k+1) \cdot |\Sigma^t| \cdot |\hat{Q}| \quad (2.1)$$

steps, where  $k$  is the sum of the current lengths of the strings on the first  $t$  tapes. During each such step, the sum of the lengths of the strings on the first  $t$  tapes increases by at most 1, since  $T$  is normalized (cf. Definition 2.1). Thus, while the direction of none of the heads on tapes  $1, \dots, t$  changes, the sum of the lengths of the strings on the first  $t$  tapes remains

$$\leq k + (k+1) \cdot |\Sigma^t| \cdot |\hat{Q}| = |\Sigma^t| \cdot |\hat{Q}| + (|\Sigma^t| \cdot |\hat{Q}| + 1) \cdot k. \quad (2.2)$$

Initially, the sum of the lengths of the strings on the first  $t$  tapes is the input length  $N$ . An easy calculation based on (2.2) shows that with at most  $i$  changes of the direction of a head on the first  $t$  tapes, the sum of the lengths of the strings on the first  $t$  tapes remains

$$\leq (N + |\Sigma^t| \cdot |\hat{Q}|) \cdot (|\Sigma^t| \cdot |\hat{Q}| + 1)^{i+1}.$$

From (2.1) one thus obtains that with at most  $i$  changes of the direction of a head on the first  $t$  tapes,  $T$  can make at most

$$N \cdot (|\Sigma^t| \cdot |\hat{Q}| + 1)^{i+2} + (|\Sigma^t| \cdot |\hat{Q}| + 1)^{i+3}$$

steps. Thus with  $r(N)$  changes of head directions, the total number of steps is at most

$$N \cdot (|\Sigma^t| \cdot |\hat{Q}| + 1)^{r(N)+2} + (|\Sigma^t| \cdot |\hat{Q}| + 1)^{r(N)+3} = N \cdot 2^{O((t+s(N)) \cdot r(N))},$$

where the constant hidden in the  $O$ -notation only depends on the parameters  $\Sigma$ ,  $Q$ ,  $u$  of the Turing machine  $T$  (and not on  $N, r, s, t$ ).  $\square$

In analogy to the definition of *randomized* complexity classes such as the class RP of randomized polynomial time (cf., e.g., [21]), we consider the randomized versions RST( $\dots$ ) and *LasVegas*-RST( $\dots$ ) of the ST( $\dots$ ) and NST( $\dots$ ) classes. The following definition of randomized Turing machines formalizes the intuition that in each step, a coin can be tossed to determine which particular successor configuration is chosen in this step. For a configuration  $\gamma$  of an NTM  $T$ , we write  $\text{Next}_T(\gamma)$  to denote the set of all configurations  $\gamma'$  that can be reached from  $\gamma$  in a single step. Each such configuration  $\gamma' \in \text{Next}_T(\gamma)$  is chosen with uniform probability, i.e.,  $\Pr(\gamma \rightarrow_T \gamma') = 1/|\text{Next}_T(\gamma)|$ . For a run  $\rho = (\rho_1, \dots, \rho_\ell)$ , the probability  $\Pr(\rho)$  that  $T$  performs run  $\rho$  is the product of the probabilities  $\Pr(\rho_i \rightarrow_T \rho_{i+1})$ , for all  $i < \ell$ . For an input word  $w$ , the probability that  $T$  accepts  $w$  (resp., that  $T$  outputs  $w'$ ) is defined as the sum of  $\Pr(\rho)$  for all accepting runs  $\rho$  of  $T$  on input  $w$  (resp., of all runs of  $T$  on  $w$  that output  $w'$ ). We say that a decision problem  $L$  is solved by a  $(\frac{1}{2}, 0)$ -RTM if, and only if, there is an NTM  $T$  such that every run of  $T$  has finite length, and the following is true for all input instances  $w$ : If  $w \in L$ , then  $\Pr(T \text{ accepts } w) \geq 1/2$ ; if  $w \notin L$ , then  $\Pr(T \text{ accepts } w) = 0$ . Similarly, we say that a function  $f : \Sigma^* \rightarrow \Sigma^*$  is computed by a *LasVegas*-RTM if, and only if, there is an NTM  $T$  such that every run of  $T$  on every input instance  $w$  has finite length and outputs either  $f(w)$  or “*I don't know*”, and  $\Pr(T \text{ outputs } f(w)) \geq 1/2$ .

**Definition 2.5 (The classes RST( $\dots$ ) and *LasVegas*-RST( $\dots$ )).**

Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ .

- (a) A decision problem  $L$  belongs to the class RST( $r, s, t$ ), if it can be solved by a  $(\frac{1}{2}, 0)$ -RTM that is  $(r, s, t)$ -bounded.
- (b) A function  $f : \Sigma^* \rightarrow \Sigma^*$  belongs to *LasVegas*-RST( $r, s, t$ ), if it can be solved by a *LasVegas*-RTM that is  $(r, s, t)$ -bounded.  $\dashv$

For classes  $R$  and  $S$  of functions we define  $\text{ST}(R, S, t) := \bigcup_{r \in R, s \in S} \text{ST}(r, s, t)$  and  $\text{ST}(R, S, O(1)) := \bigcup_{r \in \mathbb{N}} \text{ST}(R, S, t)$ . Analogous notations are used for the NST( $\dots$ ), RST( $\dots$ ), and *LasVegas*-RST( $\dots$ ) classes, too.

Let us remark that, unlike the classical complexity class ZPP, the class *LasVegas-RST*( $R, S, t$ ) is usually not equivalent to a version where the machine always gives the correct answer and has expected resource bounds  $r \in R$  and  $s \in S$ . Of course this depends on the choice of  $R$  and  $S$ .

As a straightforward observation one obtains:

**Proposition 2.6.** *For all  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ ,  $\text{ST}(r, s, t) \subseteq \text{RST}(r, s, t) \subseteq \text{NST}(r, s, t)$ .  $\dashv$*

As usual, for every (complexity) class  $C$  of decision problems,  $\text{co-}C$  denotes the class of all decision problems whose *complements* belong to  $C$ . Note that the  $\text{RST}(\dots)$ -classes consist of decision problems that can be solved by randomized algorithms with one-sided bounded error, where no false positive answers are allowed and the probability of false negative answers is at most 0.5. In contrast to this, the  $\text{co-RST}(\dots)$ -classes consist of problems that can be solved by randomized algorithms where no false negative answers are allowed and the probability of false positive answers is at most 0.5.

From Lemma 2.4, one immediately obtains for all functions  $r, s$  with  $r(N) \cdot s(N) \in O(\log N)$  that  $\text{ST}(r, s, O(1)) \subseteq \text{PTIME}$ ,  $\text{RST}(r, s, O(1)) \subseteq \text{RP}$ , and  $\text{NST}(r, s, O(1)) \subseteq \text{NP}$  (where  $\text{PTIME}$ ,  $\text{RP}$ , and  $\text{NP}$  denote the class of problems solvable in polynomial time on deterministic, randomized, and nondeterministic Turing machines, respectively).

### 3 Main Results

Our main results deal with the *sorting problem*

**SORT:**

*Instance:*  $v_1 \# \dots \# v_m \#$ , where  $m \geq 1$  and  $v_1, \dots, v_m \in \{0, 1\}^*$

*Output:*  $v_{\psi(1)} \# \dots \# v_{\psi(m)} \#$ , where  $\psi$  is a permutation of  $\{1, \dots, m\}$  such that  $v_{\psi(1)} \leq \dots \leq v_{\psi(m)}$  (with  $\leq$  denoting the lexicographic order)

and the related decision problems *checksort* and *(multi)set equality*. The *(multi)set equality problem* asks if two given (multi)sets of strings are the same. The *checksort problem* asks for two input lists of strings whether the second list is the lexicographically sorted version of the first list. Similarly as for the *sorting problem*, we encode inputs as strings over the alphabet  $\{0, 1, \#\}$ . Precisely, the input instances of each of the problems *set-equality*, *multiset-equality*, and *checksort* are

*Instance:*  $v_1 \# \dots \# v_m \# v'_1 \# \dots \# v'_m \#$ , where  $m \geq 1$  and  $v_1, \dots, v_m, v'_1, \dots, v'_m \in \{0, 1\}^*$

and the task is:

**SET-EQUALITY:**

Decide if  $\{v_1, \dots, v_m\} = \{v'_1, \dots, v'_m\}$ .

**MULTISET-EQUALITY:**

Decide if the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are equal (i.e., they contain the same elements with the same multiplicities).

**CHECK-SORT:**

Decide if  $v'_1, \dots, v'_m$  is the lexicographically sorted (in ascending order) version of  $v_1, \dots, v_m$ .

For an instance  $v_1 \# \dots \# v_m \# v'_1 \# \dots \# v'_m \#$  of the above problems, we usually let  $N = 2m + \sum_{i=1}^m (|v_i| + |v'_i|)$  denote the size of the input. Furthermore, in our proofs we will mainly consider instances where all the  $v_i$  and  $v'_i$  have the same length  $n$ , so that  $N = 2m \cdot (n + 1)$ .

Using results of Chen and Yap [9] along with a suitable implementation of the well-known *merge sort* algorithm, it is not difficult to obtain the following upper bound for the above problems:

**Proposition 3.1.** *Let  $s : \mathbb{N} \rightarrow \mathbb{N}$  be space-constructible. Then there is a function  $r : \mathbb{N} \rightarrow \mathbb{N}$  with  $r(N) \in O(\log \frac{N}{s(N)})$  such that the problem SORT, and thus each of the problems CHECK-SORT, SET-EQUALITY, MULTISSET-EQUALITY, belongs to  $\text{ST}(r(N), s(N), 2)$ .*

*Proof:* If  $s(N) \in o(\log N)$ , then  $\log(N/s(N)) = \Omega(\log N)$ , and therefore, the proposition follows directly from [9, Lemma 7] stating that  $k$  strings of arbitrary length can be sorted within  $O(\log k)$  head reversals, no internal memory space, and two external memory tapes.

If  $s(N) \in \Omega(\log N)$ , then SORT can be solved by first splitting the input instance  $v_1\#v_2\#\dots\#v_m\#$  into a sequence of *short* strings  $v_i$  (of length at most  $s(N)/2$ ) and a sequence of *long* strings  $v_i$  (of length more than  $s(N)/2$ ), then sorting both sequences separately, and finally, merging them together. Note that for the splitting we need to compute  $s(N)$ , which is possible in internal memory by the space-constructibility of  $s$ . Note also that the sequence of long strings contains at most  $k \leq 2N/s(N)$  strings, and therefore, the algorithm described in [9, Lemma 7] can be used to sort this sequence with  $O(\log k) = O(\log(N/s(N)))$  head reversals, no internal memory space, and two external memory tapes.

To sort the sequence of short strings we use an implementation of the merge sort algorithm, which, in a preprocessing step, sorts consecutive subsequences of length at most  $s(N)$  (here, the length is the total number of symbols in the sequence) using internal memory of size  $s(N)$ . In step  $i+1$ , we then have to merge sorted subsequences of length  $\leq s(N) \cdot 2^i$  into sorted subsequences of length  $\leq s(N) \cdot 2^{i+1}$ , which can be done in  $\text{ST}(O(1), s(N), 2)$  using a similar technique as described in [9, Lemma 7]. It is now easy to see that sorting the short strings is in  $\text{ST}(O(\log(N/s(N))), s(N), 2)$ .

To merge the sorted sequences of short and long strings at the end, we can once again use the technique mentioned above. Altogether, this finishes the proof of Proposition 3.1.  $\square$

Our main technical result is the following theorem which establishes a lower bound that precisely matches the upper bound of Proposition 3.1:

**Theorem 3.2.** *Let  $s : \mathbb{N} \rightarrow \mathbb{N}$  be such that  $s(N) \in o(N)$ . Furthermore, let  $r : \mathbb{N} \rightarrow \mathbb{N}$  such that  $r(N) \in o(\log \frac{N}{s(N)})$ . Then none of the problems CHECK-SORT, SET-EQUALITY, MULTISSET-EQUALITY belongs to  $\text{RST}(r(N), s(N), O(1))$ , and the problem SORT does not belong to  $\text{LasVegas-RST}(r(N), s(N), O(1))$ .*

Sections 4–7 are devoted to the proof of Theorem 3.2. The proof uses an intermediate computation model called *list machines* and proceeds by (1) showing that randomized Turing machine computations can be simulated by randomized list machines that have the same acceptance probabilities as the given Turing machines and (2) proving a lower bound for (MULTI)SET-EQUALITY and CHECK-SORT on randomized list machines.

Note that for the particular choice of  $s(N) = N/\log N$ , Proposition 3.1 and Theorem 3.2 imply that, e.g., CHECK-SORT is in  $\text{ST}(O(\log \log N), O(\frac{N}{\log N}), 2)$ , but not in  $\text{RST}(o(\log \log N), O(\frac{N}{\log N}), O(1))$ . Accordingly, when choosing  $s(N) = N^{1-\varepsilon}$ , Theorem 3.2 and Proposition 3.1 immediately lead to

**Corollary 3.3.** *Each of the problems CHECK-SORT, SET-EQUALITY, MULTISSET-EQUALITY belongs to  $\text{ST}(O(\log N), O(1), 2)$ , but for each constant  $\varepsilon$  with  $0 < \varepsilon < 1$ , none of these problems belongs to  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ .*

*Similarly, the problem SORT belongs to  $\text{ST}(O(\log N), O(1), 2)$  but, for each constant  $\varepsilon$  with  $0 < \varepsilon < 1$ , it does not belong to  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ .*  $\dashv$

In particular with communication complexity arguments in mind, one might suspect that the reason for the lower bounds is that it is hard to compare just two long strings. However, this is not so. We can prove that the problems remain hard if restricted to inputs where the length of the strings  $v_i, v'_i$  is logarithmically bounded in  $m$ . Formally, we consider the following restricted versions of the problems SORT, CHECK-SORT, and (MULTI)SET-EQUALITY:

SHORT-SORT

*Instance:*  $v_1\#\dots\#v_m\#$ , where  $m \geq 1$  and  $v_1, \dots, v_m \in \{0, 1\}^*$  such that  $|v_i| \leq 2 \cdot \log m$  for  $1 \leq i \leq m$ .

*Output:*  $v_{\psi(1)}\#\dots\#v_{\psi(m)}\#$ , where  $\psi$  is a permutation of  $\{1, \dots, m\}$  such that  $v_{\psi(1)} \leq \dots \leq v_{\psi(m)}$  (with  $\leq$  denoting the lexicographic order)

Similarly, if  $P$  is one of the problems CHECK-SORT, SET-EQUALITY, and (MULTI)SET-EQUALITY, the problem SHORT- $P$  is defined as follows:

SHORT- $P$

*Instance:*  $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$ , where  $m \geq 1$  and  $v_1, \dots, v_m, v'_1, \dots, v'_m \in \{0, 1\}^*$  such that each  $v_i$  and  $v'_i$  is a 0-1-string of length at most  $2 \cdot \log m$

*Problem:* Decide if the input is a “yes”-instance for the problem  $P$ .

By applying a suitable reduction, we obtain that the bound of Corollary 3.3 even applies to the “SHORT-” versions of SORT, CHECK-SORT, and (MULTI)SET-EQUALITY:

**Theorem 3.4.** *Let  $\varepsilon$  be a constant with  $0 < \varepsilon < 1$ . Then, none of the problems SHORT-SET-EQUALITY, SHORT-MULTISET-EQUALITY, and SHORT-CHECK-SORT belongs to  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ . Similarly, the problem SHORT-SORT does not belong to  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ .  $\dashv$*

The proof of Theorem 3.4 is deferred to Section 7. As a further result we show that:

**Theorem 3.5.**

(a)  $\text{MULTISET-EQUALITY}$  belongs to  $\text{co-RST}(2, O(\log N), 1) \subseteq \text{co-NST}(2, O(\log N), 1)$ .

(b) Each of the problems  $\text{MULTISET-EQUALITY}$ ,  $\text{CHECK-SORT}$ ,  $\text{SET-EQUALITY}$  belongs to  $\text{NST}(3, O(\log N), 2)$ .  $\dashv$

*Proof:* (a): We apply fairly standard *fingerprinting techniques* and show how to implement them on a  $(2, O(\log N), 1)$ -bounded randomized Turing machine. Consider an instance  $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$  of the  $\text{MULTISET-EQUALITY}$  problem. For simplicity, let us assume that all the  $v_i$  and  $v'_j$  have the same length  $n$ . Thus the input size  $N$  is  $2 \cdot m \cdot (n + 1)$ . We view the  $v_i$  and  $v'_i$  as integers in  $\{0, \dots, 2^n - 1\}$  represented in binary.

We use the following algorithm to decide whether the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are equal:

- (1) During a first sequential scan of the input, determine the input parameters  $n$ ,  $m$ , and  $N$ .
- (2) Choose a prime  $p_1 \leq k := m^3 \cdot n \cdot \log(m^3 \cdot n)$  uniformly at random.
- (3) Choose an arbitrary prime  $p_2$  such that  $3k < p_2 \leq 6k$ . Such a prime exists by Bertrand’s postulate.
- (4) Choose  $x \in \{1, \dots, p_2 - 1\}$  uniformly at random.
- (5) For  $1 \leq i \leq m$ , let  $e_i = (v_i \bmod p_1)$  and  $e'_i = (v'_i \bmod p_1)$ . If

$$\sum_{i=1}^m x^{e_i} \equiv \sum_{i=1}^m x^{e'_i} \pmod{p_2} \quad (3.1)$$

then accept, else reject.

Let us first argue that the algorithm is correct (for sufficiently large  $m, n$ ): Clearly, if the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are equal then the algorithm accepts. In the following, we estimate the probability that the algorithm fails if the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are distinct. So assume that  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are distinct. Then the probability that the multisets  $\{e_1, \dots, e_m\}$  and  $\{e'_1, \dots, e'_m\}$  are equal is  $O(1/m)$ . This follows from the following claim.

**Claim 1** *Let  $n, m \in \mathbb{N}$ ,  $k = m^3 \cdot n \cdot \log(m^3 \cdot n)$ , and  $0 \leq v_1, \dots, v_m, v'_1, \dots, v'_m < 2^n$ . Then for a prime  $p \leq k$  chosen uniformly at random,  $\Pr(\exists i, j \leq m$  with  $v_i \neq v'_j$  and  $v_i \equiv v'_j \pmod{p}) \leq O(1/m)$ .*

*Proof:* We use the following well-known result (see, for example, Theorem 7.5 of [18]): Let  $n, \ell \in \mathbb{N}$ ,  $k = \ell \cdot n \cdot \log(\ell \cdot n)$ , and  $0 < x < 2^n$ . Then for a prime  $p \leq k$  chosen uniformly at random,

$$\Pr(x \equiv 0 \pmod{p}) \leq O\left(\frac{1}{\ell}\right).$$

The claim then follows if we apply this result with  $\ell = m^3$  simultaneously to the at most  $m^2$  numbers  $x = v_i - v'_j$  with  $v_i \neq v'_j$ .  $\square$

To proceed with the proof of Theorem 3.5(a), suppose that the two multisets  $\{e_1, \dots, e_m\}$  and  $\{e'_1, \dots, e'_m\}$  are distinct. Then the polynomial

$$q(X) = \sum_{i=1}^m X^{e_i} - \sum_{i=1}^m X^{e'_i}$$

is nonzero. Note that all coefficients and the degree of  $q(X)$  are at most  $k < p_2$ . We view  $q(X)$  as a polynomial over the field  $\mathbb{F}_{p_2}$ . As a nonzero polynomial of degree at most  $p_1$ , it has at most  $p_1$  zeroes. Thus the probability that  $q(x) = 0$  for the randomly chosen  $x \in \{1, \dots, p_2 - 1\}$  is at most  $p_1 / (p_2 - 1) \leq 1/3$ . Therefore, if the multisets  $\{e_1, \dots, e_m\}$  and  $\{e'_1, \dots, e'_m\}$  are distinct, the algorithm accepts with probability at most  $1/3$ . Thus, if the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are distinct, then the probability that the algorithm accepts (i.e., fails) is at most

$$O\left(\frac{1}{m}\right) + \frac{1}{3} \leq \frac{1}{2}$$

for sufficiently large  $m$ . This proves the correctness of the algorithm.

Let us now explain how to implement the algorithm on a  $(2, O(\log N), 1)$ -bounded randomized Turing machine. Note that the binary representations of the primes  $p_1$  and  $p_2$  have length  $O(\log N)$ . The standard arithmetical operations can be carried out in linear space on a Turing machine. Thus with numbers of length  $O(\log N)$ , we can carry out the necessary arithmetic on the internal memory tapes of our  $(2, O(\log N), 1)$ -bounded Turing machine.

To choose a random prime  $p_1$  in step (2), we simply choose a random number  $\leq k$  and then test if it is prime, which is easy in linear space. If the number is not prime, we repeat the procedure, and if we do this sufficiently often, we can find a random prime with high probability. Steps (3) and (4) can easily be carried out in internal memory. To compute the number  $e_i$  in step (5), we proceed as follows: Suppose the binary representation of  $v_i$  is  $v_{i,(n-1)} \dots v_{i,0}$ , where  $v_{i,0}$  is the least significant bit. Observe that

$$e_i = \left( \left( \sum_{j=0}^{n-1} 2^j \cdot v_{i,j} \right) \bmod p_1 \right).$$

We can evaluate this sum sequentially by taking all terms modulo  $p_1$ ; this way we only have to store numbers smaller than  $p_1$ . This requires one sequential scan of  $v_i$  and no head reversals.

To evaluate the polynomial  $\sum_{i=1}^m x^{e_i}$  modulo  $p_2$ , we proceed as follows: Let  $t_i = (x^{e_i} \bmod p_1)$  and  $s_i = ((\sum_{j=1}^i t_j) \bmod p_1)$ . Again we can compute the sum sequentially by computing  $e_i$ ,  $t_i$ , and  $s_i = ((s_{i-1} + t_i) \bmod p_1)$  for  $i = 1, \dots, m$ . We can evaluate  $\sum_{i=1}^m x^{e_i}$  analogously and then test if (3.1) holds. This completes the proof of part (a) of Theorem 3.5.

(b): Let  $w$  be an input of length  $N$ ,  $w := v_1 \# v_2 \# \dots \# v_m \# v'_1 \# v'_2 \# \dots \# v'_m \#$ . Note that the multisets  $\{v_1, \dots, v_m\}$  and  $\{v'_1, \dots, v'_m\}$  are equal if and only if there is a permutation  $\pi$  of  $\{1, \dots, m\}$  such that for all  $i \in \{1, \dots, m\}$ ,  $v_i = v'_{\pi(i)}$ . The idea is to “guess” such a permutation  $\pi$  (suitably encoded as a string over  $\{0, 1, \#\}$ ), to write sufficiently many copies of the string  $u := \pi \# w$  onto the first tape, and finally solve the problem by comparing  $v_i$  and  $v'_{\pi(i)}$  bitwise, where in each step we use the next copy of  $u$ .

A  $(3, O(\log N), 2)$ -bounded nondeterministic Turing machine  $M$  can do this as follows. In a forward scan, it nondeterministically writes a sequence  $u_1, u_2, \dots, u_\ell$  of  $\ell := N \cdot m + m$  many strings on its first and on its second tape, where

$$u_i := \pi_{i,1} \# \dots \# \pi_{i,m} \# v_{i,1} \# \dots \# v_{i,m} \# v'_{i,1} \# \dots \# v'_{i,m} \#,$$

$\pi_{i,j}$  is the binary encoding of a number in  $\{1, \dots, m\}$ , and  $v_{i,j}$  and  $v'_{i,j}$  are bit strings of length at most  $N$ . For each  $i \in \{1, \dots, \ell\}$ , the substring  $\pi_{i,1} \# \dots \# \pi_{i,m} \#$  of  $u_i$  is intended to be an encoding of a permutation  $\pi$  of  $\{1, \dots, m\}$ , and the substring  $v_{i,1} \# \dots \# v_{i,m} \# v'_{i,1} \# \dots \# v'_{i,m} \#$  is intended to be a copy of the input string  $w$  (i.e.,  $v_{i,j} = v_j$  and  $v'_{i,j} = v'_j$  for all  $j \in \{1, \dots, m\}$ ).

While writing the first  $N \cdot m$  strings,  $M$  ensures that for every  $i \in \{1, \dots, N \cdot m\}$ , either  $v_{i, \lceil i/N \rceil}$  and  $v'_{i, \pi_{i, \lceil i/N \rceil}}$  coincide on position  $((i-1) \bmod N) + 1$ , or that the length of both strings is less than  $((i-$

1) mod  $N$ ) + 1. While writing the last  $m$  strings,  $M$  ensures that for all  $i \in \{1, \dots, m\}$  and  $j \in \{i + 1, \dots, m\}$ ,  $\pi_{N-m+i,i} \neq \pi_{N-m+i,j}$ . Finally,  $M$  checks in a backward scan of both external memory tapes that  $u_i = u_{i-1}$  for all  $i \in \{2, \dots, \ell\}$ , and that  $v_{1,j} = v_j$  and  $v'_{1,j} = v'_j$  for all  $j \in \{1, \dots, m\}$ .

The SET-EQUALITY problem can be solved in a similar way. Furthermore, to decide CHECK-SORT the machine additionally has to check that  $v'_i$  is smaller than or equal to  $v'_{i+1}$  for all  $i \in \{1, \dots, m-1\}$ . This can be done, e.g., by writing  $n \cdot \sum_{i=1}^{m-1} i$  additional copies of  $u$ , and by comparing  $v'_i$  and  $v'_{i+1}$  bitwise on these strings for each  $i$ .

This finally completes the proof of Theorem 3.5.  $\square$

Proposition 2.6, Corollary 3.3 and Theorem 3.5 immediately lead to the following separations between the deterministic, randomized, and nondeterministic  $\text{ST}(\dots)$  classes:

**Corollary 3.6.** *Let  $\varepsilon$  be a constant with  $0 < \varepsilon < 1$ . Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$  such that  $r(N) \in o(\log N)$  and  $s(N) \in O(N^{1-\varepsilon}) \cap \Omega(\log N)$ . Then,*

- (a)  $\text{RST}(O(r), O(s), O(1)) \neq \text{co-RST}(O(r), O(s), O(1))$ , and
- (b)  $\text{ST}(O(r), O(s), O(1)) \subsetneq \text{RST}(O(r), O(s), O(1)) \subsetneq \text{NST}(O(r), O(s), O(1))$ .

*Proof:* (a) is an immediate consequence of Corollary 3.3 and Theorem 3.5 (a).

The inclusions  $\text{ST}(O(r), O(s), O(1)) \subseteq \text{RST}(O(r), O(s), O(1)) \subseteq \text{NST}(O(r), O(s), O(1))$  in (b) follow directly from Proposition 2.6. The second inequality in (b) follows directly from Corollary 3.3 and Theorem 3.5 (b). The first inequality in (b) holds because, due to Theorem 3.5 (a), the complement of the MULTISSET-EQUALITY problem belongs to  $\text{RST}(2, O(\log N), 1)$ . Since the deterministic  $\text{ST}(\dots)$  classes are closed under taking complements, Corollary 3.3 implies that the complement of MULTISSET-EQUALITY does not belong to  $\text{ST}(O(r), O(s), O(1))$ .  $\square$

### 3.1 Lower Bounds for Query Evaluation

Our lower bound of Corollary 3.3 for the SET-EQUALITY problem leads to the following lower bounds on the worst case data complexity of database query evaluation problems in a streaming context:

**Theorem 3.7 (Bounds for Relational Algebra).**

- (a) *For every relational algebra query  $Q$ , the problem of evaluating  $Q$  on a stream consisting of the tuples of the input database relations can be solved in  $\text{ST}(O(\log N), O(1), O(1))$ .*
- (b) *There exists a relational algebra query  $Q'$  such that the problem of evaluating  $Q'$  on a stream of the tuples of the input database relations is not in  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for any constant  $\varepsilon$  with  $0 < \varepsilon < 1$ .*  $\dashv$

*Proof:* (a): It is straightforward to see that for every relational algebra query  $Q$  there exists a number  $c_Q$  such that  $Q$  can be evaluated within  $c_Q$  sequential scans and sorting steps. Every sequential scan accounts for a constant number of head reversals and constant internal memory space. Each sorting step can be accomplished using the sorting method of [9, Lemma 7] (which is a variant of the merge sort algorithm) with  $O(\log N)$  head reversals and constant internal memory space. Since the number  $c_Q$  of necessary sorting steps and scans is constant (i.e., only depends on the query, but not on the input size  $N$ ), the query  $Q$  can be evaluated by an  $(O(\log N), O(1), O(1))$ -bounded deterministic Turing machine.

(b): Consider the relational algebra query

$$Q' := (R_1 - R_2) \cup (R_2 - R_1)$$

which computes the symmetric difference of two relations  $R_1$  and  $R_2$ . Note that the query result is empty if, and only if,  $R_1 = R_2$ . Therefore, any algorithm that evaluates  $Q'$  solves, in particular, the SET-EQUALITY problem. Hence, if  $Q'$  could be evaluated in  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for some constant  $\varepsilon$  with  $0 < \varepsilon < 1$ , then SET-EQUALITY could be solved in  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ , contradicting Corollary 3.3.  $\square$

We also obtain lower bounds on the worst case data complexity of evaluating  $XQuery$  and  $XPath$  queries against XML document streams:

**Theorem 3.8 (Lower Bounds for XQuery and XPath).** (a) *There is an XQuery query  $Q$  such that the problem of evaluating  $Q$  on an input XML document stream of length  $N$  does not belong to the class  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for any constant  $\varepsilon$  with  $0 < \varepsilon < 1$ .*

(b) *There is an XPath query  $Q$  such that the problem of filtering an input XML document stream with  $Q$  (i.e., checking whether at least one node of the document matches the query) does not belong to the class  $\text{co-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for any constant  $\varepsilon$  with  $0 < \varepsilon < 1$ .  $\dashv$*

*Proof:* We represent an instance  $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$  of the SET-EQUALITY problem by an XML document of the form

```
<instance>
  <set1>
    <item> <string>  $x_1$  </string> </item>
    ...
    <item> <string>  $x_m$  </string> </item>
  </set1>
  <set2>
    <item> <string>  $y_1$  </string> </item>
    ...
    <item> <string>  $y_m$  </string> </item>
  </set2>
</instance>
```

For technical reasons, we enclose every string  $x_i$  and  $y_j$  by a `string`-element *and* an `item`-element. For the proof of part (a) of the theorem, one of the two would suffice, but for the proof of part (b) it is more convenient if each  $x_i$  and  $y_j$  is enclosed by two element nodes.

It should be clear that, given as input  $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$ , the above XML document can be produced by using a constant number of sequential scans, constant internal memory space, and two external memory tapes.

To prove (a), we express the SET-EQUALITY problem by the following XQuery query  $Q$

```
<result>
  if ( every $x in /instance/set1/item/string satisfies
        some $y in /instance/set2/item/string satisfies
        $x = $y )
    and
    ( every $y in /instance/set2/item/string satisfies
        some $x in /instance/set1/item/string satisfies
        $x = $y )
  then <true/>
  else ()
</result>
```

Note that if  $\{x_1, \dots, x_m\} = \{y_1, \dots, y_m\}$ , then  $Q$  returns the document `<result><true/></result>`, and otherwise  $Q$  returns the “empty” document `<result></result>`. Thus, if  $Q$  could be evaluated in  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for some constant  $\varepsilon$  with  $0 < \varepsilon < 1$ , then the SET-EQUALITY problem could be solved in  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ , contradicting Corollary 3.3.

To prove (b), consider the following XPath query  $Q$ :

```
descendant::set1/child::item[not child::string =
  ancestor::instance/child::set2/child::item/child::string]
```

It selects all `item`-nodes below `set1` whose string content does *not* occur as the string content of some `item`-node below `set2` (recall the “existential” semantics of XPath [25]). In other words:  $Q$  selects all (nodes that represent) elements in  $X - Y$ , for  $X := \{x_1, \dots, x_m\}$  and  $Y := \{y_1, \dots, y_m\}$ .

Now assume, for contradiction, that the problem of filtering an input XML document stream with the XPath query  $Q$  (i.e., checking whether at least one document node is selected by  $Q$ ) belongs to the class  $\text{co-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$  for some constant  $\varepsilon$  with  $0 < \varepsilon < 1$ . Then, clearly, there exists an

$(o(\log N), O(N^{1-\epsilon}), O(1))$ -bounded randomized Turing machine  $T$  which has the following properties for every input  $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$  (where  $X := \{x_1, \dots, x_m\}$  and  $Y := \{y_1, \dots, y_m\}$ ):

- (1) If  $Q$  selects at least one node (i.e.,  $X - Y \neq \emptyset$ , i.e.,  $X \not\subseteq Y$ ), then  $T$  accepts with probability 1.
- (2) If  $Q$  does not select any node (i.e.,  $X - Y = \emptyset$ , i.e.,  $X \subseteq Y$ ), then  $T$  rejects with probability  $\geq 0.5$ .

By running this machine for the input  $x_1\#\dots\#x_m\#y_1\#\dots\#y_m\#$ , and another time for  $y_1\#\dots\#y_m\#x_1\#\dots\#x_m\#$ , we can easily construct a randomized Turing machine  $\tilde{T}$  witnessing that the SET-EQUALITY problem belongs to  $\text{RST}(o(\log N), O(N^{1-\epsilon}), O(1))$ . This yields the desired contradiction.  $\square$

## 4 List Machines

This section as well as the subsequent sections are devoted to the proof of Theorem 3.2. For proving Theorem 3.2 we use *list machines*. The important advantage that these list machines have over the original Turing machines is that they make it fairly easy to track the “flow of information” during a computation.

### Definition 4.1 (Nondeterministic List Machine).

A *nondeterministic list machine (NLM)* is a tuple

$$M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$$

consisting of

- a  $t \in \mathbb{N}$ , the *number of lists*.
- an  $m \in \mathbb{N}$ , the *length of the input*.
- a finite set  $I$  whose elements are called *input numbers* (usually,  $I \subseteq \mathbb{N}$  or  $I \subseteq \{0, 1\}^*$ ).
- a finite set  $C$  whose elements are called *nondeterministic choices*.
- a finite set  $A$  whose elements are called (*abstract*) *states*.

We assume that  $I$ ,  $C$ , and  $A$  are pairwise disjoint and do not contain the two special symbols ‘ $\langle$ ’ and ‘ $\rangle$ ’.

We call  $\mathbb{A} := I \cup C \cup A \cup \{\langle, \rangle\}$  the *alphabet* of the machine, and  $\mathbf{A} := \mathbb{A}^*$  the set of potential entries in a list cell.

- an *initial state*  $a_0 \in A$ .
- a *transition function*

$$\alpha : (A \setminus B) \times (\mathbf{A})^t \times C \rightarrow (A \times \text{Movement}^t)$$

with

$$\text{Movement} := \{ (\text{head-direction}, \text{move}) \mid \text{head-direction} \in \{-1, +1\}, \text{move} \in \{\text{true}, \text{false}\} \}.$$

- a set  $B \subseteq A$  of *final states*.
- a set  $B_{acc} \subseteq B$  of *accepting states*. (We use  $B_{rej} := B \setminus B_{acc}$  to denote the set of *rejecting* states.)  $\dashv$

Intuitively, an NLM  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  operates as follows: The input is a sequence  $(v_1, \dots, v_m) \in I^m$ . Instead of tapes (as a Turing machine), an NLM operates on  $t$  lists. In particular, this means that a new list cell can be inserted between two existing cells. As for tapes, there is a read-write head operating on each list. Cells of the lists store strings in  $\mathbb{A}^*$ , i.e., elements in  $\mathbf{A}$  (and not just symbols from  $\mathbb{A}$ ). Initially, the first list, called the *input list*, contains  $(v_1, \dots, v_m)$  (i.e. the  $i$ th list cell contains entry  $v_i$ ), and all other lists just consist of one dummy element. The heads are on the left end of the lists. It is crucial that the transition function of a list machine only determines the machine’s next state and its head movements, but *not what is written into the list cells*. In each step of the computation, the heads move according to the transition function, by choosing “nondeterministically” an arbitrary element in  $C$ . Furthermore, in each step, the current state, the content of the list cells at all current head positions,

and the nondeterministic choice  $c \in C$  used in the current transition, are written on each list *behind* the head. Here “behind” is defined with respect to the current direction of the head. When a final state is reached, the machine stops. If this final state belongs to  $B_{acc}$ , the according run is *accepting*; otherwise it is *rejecting*. Figure 1 illustrates a transition of an NLM.

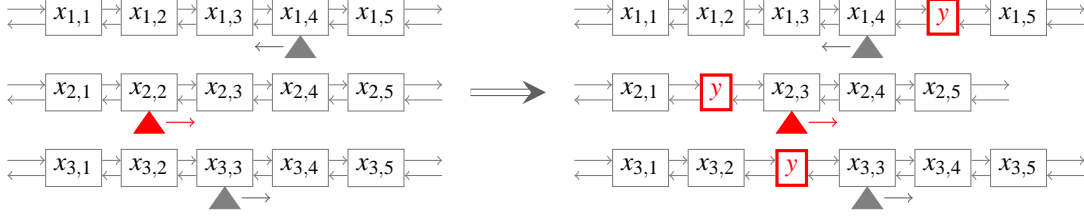


Figure 1: A transition of an NLM. The example transition is of the form  $(a, x_{1,4}, x_{2,2}, x_{3,3}, c) \rightarrow (b, (-1, false), (1, true), (1, false))$ . The new string  $y = \langle a \ x_{1,4} \ x_{2,2} \ x_{3,3} \ c \rangle$  that is written in the tape cells consists of the current state  $a$ , the content of the list cells read before the transition, and the nondeterministic choice  $c$ .

Formally, the semantics of nondeterministic list machines are defined as follows:

**Definition 4.2 (Semantics of NLMs).**

Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM.

(a) A *configuration* of  $M$  is a tuple  $(a, \bar{p}, \bar{d}, X)$  with

$$a \in A, \quad \bar{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_t \end{pmatrix} \in \mathbb{N}^t, \quad \bar{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_t \end{pmatrix} \in \{-1, +1\}^t, \quad X = \begin{pmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_t \end{pmatrix} \in (\mathbf{A}^*)^t,$$

where

- $a$  is the *current state*,
- $\bar{p}$  is the tuple of *head positions*,
- $\bar{d}$  is the tuple of *head directions*, and
- $\bar{x}_i = (x_{i,1}, \dots, x_{i,m_i}) \in \mathbf{A}^{m_i}$  for  $1 \leq i \leq t$  and some  $m_i \geq 1$  contains the *content of all the cells of the  $i$ th list*. (The string  $x_{i,j} \in \mathbf{A} = \mathbf{A}^*$  is the content of the  $j$ th cell of the  $i$ th list.)

(b) The *initial configuration* for input  $(v_1, \dots, v_m) \in I^m$  is the tuple  $(a, \bar{p}, \bar{d}, X)$ , where  $a = a_0$ ,  $\bar{p} = (1, \dots, 1)^\top$ ,  $\bar{d} = (+1, \dots, +1)^\top$ , and  $X = (\bar{x}_1, \dots, \bar{x}_t)^\top$  with

$$\bar{x}_1 = (\langle v_1 \rangle, \dots, \langle v_m \rangle) \in \mathbf{A}^m$$

and  $\bar{x}_2 = \dots = \bar{x}_t = (\langle \rangle) \in \mathbf{A}^1$ .

(c) For a nondeterministic choice  $c \in C$ , the  $c$ -*successor* of a configuration  $(a, \bar{p}, \bar{d}, X)$  is the configuration  $(a', \bar{p}', \bar{d}', X')$  defined as follows: Suppose that

$$\alpha(a, x_{1,p_1}, \dots, x_{t,p_t}, c) = (b, e_1, \dots, e_t).$$

We let  $a' = b$ . For  $1 \leq i \leq t$ , let  $m_i$  be the length of the list  $\bar{x}_i$ , and let

$$e'_i := (\text{head-direction}_i, \text{move}_i) := \begin{cases} (-1, false) & \text{if } p_i = 1 \text{ and } e_i = (-1, true), \\ (+1, false) & \text{if } p_i = m_i \text{ and } e_i = (+1, true), \\ e_i & \text{otherwise.} \end{cases}$$

This will prevent the machine from “falling off” the left or right end of a list. I.e., if the head is standing on the rightmost (resp., leftmost) list cell, it will stay there instead of moving a further step to the right (resp., to the left).

We fix  $f_i \in \{0, 1\}$  such that  $f_i = 1$  iff ( $move_i = true$  or  $head-direction_i \neq d_i$ ).

If  $f_i = 0$  for all  $i \in \{1, \dots, t\}$ , then we let  $\bar{p}' := \bar{p}$ ,  $\bar{d}' := \bar{d}$ , and  $X' := X$  (i.e., if none of the machine’s head moves or changes its direction, then the *state* is the only thing that may change in the machine’s current step).

So suppose that there is at least one  $i$  such that  $f_i \neq 0$ . In this case, we let

$$y := \langle a \langle x_{1,p_1} \rangle \cdots \langle x_{t,p_t} \rangle c \rangle.$$

For all  $i \in \{1, \dots, t\}$ , we let

$$\bar{x}'_i := \begin{cases} (x_{i,1}, \dots, x_{i,p_i-1}, y, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } move_i = true, \\ (x_{i,1}, \dots, x_{i,p_i-1}, y, x_{i,p_i}, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } d_i = +1 \text{ and } move_i = false, \\ (x_{i,1}, \dots, x_{i,p_i-1}, x_{i,p_i}, y, x_{i,p_i+1}, \dots, x_{i,m_i}) & \text{if } d_i = -1 \text{ and } move_i = false. \end{cases}$$

Furthermore, for all  $i \in \{1, \dots, t\}$ , we let

$$p'_i := \begin{cases} p_i + 1 & \text{if } e'_i = (+1, true), \\ p_i - 1 & \text{if } e'_i = (-1, true), \\ p_i + 1 & \text{if } e'_i = (+1, false), \\ p_i & \text{if } e'_i = (-1, false), \end{cases}$$

and  $d'_i := head-direction_i$ . See Figure 1 for an example of a transition from one configuration to a successor configuration.

(d) A configuration  $(a, \bar{p}, \bar{d}, X)$  is *final (accepting, resp., rejecting)*, if  $a \in B$  ( $B_{acc}$ , resp.,  $B_{rej} := B \setminus B_{acc}$ ).

A (finite) *run* of the machine is a sequence  $(\rho_1, \dots, \rho_\ell)$  of configurations, where  $\rho_1$  is the initial configuration for some input,  $\rho_\ell$  is final, and for every  $i < \ell$  there is a nondeterministic choice  $c_i \in C$  such that  $\rho_{i+1}$  is the  $c_i$ -successor of  $\rho_i$ .

A run is called *accepting (resp., rejecting)* if its final configuration is accepting (resp., rejecting).

(e) An input  $(v_1, \dots, v_m) \in I^m$  is *accepted* by machine  $M$  if there is at least one accepting run of  $M$  on input  $(v_1, \dots, v_m)$ . ⊣

For every run  $\rho$  of an NLM  $M$  and for each list  $\tau$  of  $M$ , we define  $rev(\rho, \tau)$  to be the number of changes of the direction of the  $\tau$ -th list’s head in run  $\rho$ . More precisely, let  $\rho = (\rho_1, \dots, \rho_\ell)$ . Then we say that the head of list  $\tau$  *changes its direction* in step  $i$  of  $\rho$  if and only if  $\rho_i$  has the form  $(a, \bar{p}, \bar{d}, X)$  with  $\bar{d} = (d_1, \dots, d_t)$ ,  $\rho_{i+1}$  has the form  $(a', \bar{p}', \bar{d}', X')$  with  $\bar{d}' = (d'_1, \dots, d'_t)$ , and  $d_\tau \neq d'_\tau$ . In particular,  $rev(\rho, \tau) = \sum_{i=1}^{\ell-1} rev_i(\rho, \tau)$ , where for all  $i \in \{1, \dots, \ell-1\}$ ,  $rev_i(\rho, \tau) := 1$  if the head of list  $\tau$  changes its direction in step  $i$  of  $\rho$ , and  $rev_i(\rho, \tau) := 0$  otherwise.

We say that  $M$  is *(r,t)-bounded*, for some  $r, t \in \mathbb{N}$ , if it has at most  $t$  lists, every run  $\rho$  of  $M$  is *finite*, and  $1 + \sum_{\tau=1}^t rev(\rho, \tau) \leq r$ .

An NLM is called *deterministic* if  $|C| = 1$ . *Randomized* list machines are defined in a similar way as randomized Turing machines: For configurations  $\gamma$  and  $\gamma'$  of an NLM  $M$ , the probability  $\Pr(\gamma \rightarrow_M \gamma')$  that  $\gamma$  yields  $\gamma'$  in one step, is defined as  $|\{c \in C : \gamma' \text{ is the } c\text{-successor of } \gamma\}|/|C|$ . For a run  $\rho = (\rho_1, \dots, \rho_\ell)$ , the probability  $\Pr(\rho)$  that  $M$  performs run  $\rho$  is the product of the probabilities  $\Pr(\rho_i \rightarrow_M \rho_{i+1})$ , for all  $i < \ell$ . For an input  $\bar{v} \in I^m$ , the probability that  $M$  accepts  $\bar{v}$  is defined as the sum of  $\Pr(\rho)$  for all accepting runs  $\rho$  of  $M$  on input  $\bar{v}$ .

The following notation will be very convenient:

**Definition 4.3** ( $\rho_M(\bar{v}, \bar{c})$ ). Let  $M$  be an NLM and let  $\ell \in \mathbb{N}$  such that every run of  $M$  has length  $\leq \ell$ . For every input  $\bar{v} \in I^m$  and every sequence  $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$ , we use  $\rho_M(\bar{v}, \bar{c})$  to denote the run  $(\rho_1, \dots, \rho_\ell)$  obtained by starting  $M$  with input  $\bar{v}$  and by making in its  $i$ -th step the nondeterministic choice  $c_i$  (i.e.,  $\rho_{i+1}$  is the  $c_i$ -successor of  $\rho_i$ ). ⊣

It is straightforward to see that

**Lemma 4.4.** *Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM, and let  $\ell$  be an upper bound on the length of  $M$ 's runs.*

(a) *For every run  $\rho$  of  $M$  on an input  $\bar{v} \in I^m$ , we have*

$$\Pr(\rho) = \frac{|\{\bar{c} \in C^\ell : \rho_M(\bar{v}, \bar{c}) = \rho\}|}{|C^\ell|}.$$

(b)

$$\Pr(M \text{ accepts } \bar{v}) = \frac{|\{\bar{c} \in C^\ell : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}|}{|C^\ell|}.$$

## 4.1 Basic properties of list machines

**Definition 4.5.** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an  $(r, t)$ -bounded NLM.

- (a) The *total list length* of a configuration of  $M$  is the sum of the lengths (i.e., number of cells) of all lists in that configuration.
- (b) The *cell size* of a configuration of  $M$  is defined as the maximum length of the entries of the cells occurring in the configuration (remember that the cell entries are strings over  $\mathbb{A} = I \cup C \cup A \cup \{ \langle, \rangle \}$ ).

Observe that neither the total list length nor the list size can ever decrease during a computation.

**Lemma 4.6 (List length and cell size).**

*Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an  $(r, t)$ -bounded NLM.*

- (a) *For every  $i \in \{1, \dots, r\}$ , the total list length of each configuration that occurs before the  $i$ -th change of a head direction is  $\leq (t+1)^i \cdot m$ .  
In particular, the total list length of each configuration in each run of  $M$  is  $\leq (t+1)^r \cdot m$ .*
- (b) *The cell size of each configuration in each run of  $M$  is  $\leq 11 \cdot (\max\{t, 2\})^r$ .*

*Proof:* We first prove (a). Let  $\gamma$  be a configuration of total list length  $\ell$ . Then the total list length of a successor configuration  $\gamma'$  of  $\gamma$  is at most  $\ell + t$ , if a head moves or changes its direction in the transition from  $\gamma$  to  $\gamma'$ , and it remains  $\ell$  otherwise.

Now suppose  $\gamma'$  is a configuration that can be reached from  $\gamma$  without changing the direction of any head. Then  $\gamma'$  is reached from  $\gamma$  with at most  $\ell - t$  head movements, because a head can move into the same direction for at most  $\lambda - 1$  times on a list of length  $\lambda$ . Thus the total list length of  $\gamma'$  is at most

$$\ell + t \cdot (\ell - t). \tag{4.1}$$

The total list length of the initial configuration is  $m + t - 1$ . A simple induction based on (4.1) shows that the total list length of a configuration that occurs before the  $i$ -th change of a head direction is at most

$$(t+1)^i \cdot m.$$

This proves (a).

For the proof of (b), let  $\gamma$  be a configuration of cell size  $s$ . Then the cell size of all configurations that can be reached from  $\gamma$  without changing the direction of any head is at most

$$2 + t \cdot (2 + s) + 2 = 4 + t \cdot (2 + s).$$

The cell size of the initial configuration is 3. A simple induction shows that the total cell size of any configuration that occurs before the  $i$ -th change of a head direction is at most

$$4 + \sum_{j=1}^{i-1} 6t^j + 5t^i \leq 11 \cdot (\max\{t, 2\})^i.$$

□

**Definition 4.7 (moves( $\rho$ )).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM. Let  $\rho = (\rho_1, \dots, \rho_\ell)$  be a run of  $M$ . We define

$$\text{moves}(\rho) := (\text{move}_1, \dots, \text{move}_{\ell-1}) \in (\{0, 1, -1\}^t)^{\ell-1},$$

where, for every  $i < \ell$ ,  $\text{move}_i = (\text{move}_{i,1}, \dots, \text{move}_{i,t})^\top \in \{0, 1, -1\}^t$  such that, for each  $\tau \in \{1, \dots, t\}$ ,  $\text{move}_{i,\tau} = 0$  (resp., 1, resp., -1) if, and only if, in the transition from configuration  $\rho_i$  to configuration  $\rho_{i+1}$ , the head on the  $\tau$ -th list stayed on the same list cell (resp., moved to the next cell to the right, resp., to the left).

Recall that each run of an  $(r, t)$ -bounded NLM is *finite*.

**Lemma 4.8 (The shape of runs of an NLM).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an  $(r, t)$ -bounded NLM, and let  $k := |A|$ . The following is true for every run  $\rho = (\rho_1, \dots, \rho_\ell)$  of  $M$  and the corresponding sequence  $\text{moves}(\rho) = (\text{move}_1, \dots, \text{move}_{\ell-1})$ .

(a)  $\ell \leq k + k \cdot (t+1)^{r+1} \cdot m.$

(b) There is a number  $\mu \leq (t+1)^{r+1} \cdot m$  and there are indices  $1 \leq j_1 < j_2 < \dots < j_\mu < \ell$  such that:

(i) For every  $i \in \{1, \dots, \ell-1\}$ ,

$$\text{move}_i \neq (0, 0, \dots, 0)^\top \iff i \in \{j_1, \dots, j_\mu\}.$$

I.e.,  $\mu$  is the number of steps in  $\rho$ , where at least one head is moved.

(ii) If  $\mu = 0$ , then  $\ell \leq k$ .

If  $\mu \neq 0$ , then

- $j_1 \leq k$ ,
- $j_{v+1} - j_v \leq k$ , for every  $v \in \{1, \dots, \mu-1\}$  and
- $\ell - j_\mu \leq k$ .

*Proof:* For indices  $i < \ell$  with  $\text{move}_i = (0, 0, \dots, 0)^\top$  we know from Definition 4.2(c) that the *state* is the only thing in which  $\rho_i$  and  $\rho_{i+1}$  differ. As  $(r, t)$ -bounded NLMs are *not* allowed to have an *infinite* run, we obtain that without moving any of its heads,  $M$  can make at most  $k$  consecutive steps.

On the other hand, for every  $i \in \{1, \dots, r\}$  we know from Lemma 4.6(a) that the total list length of a configuration that occurs before the  $i$ -th change of a head direction is

$$\leq (t+1)^i \cdot m. \tag{4.2}$$

Thus, between the  $(i-1)$ -st and the  $i$ -th change of a head direction, the number of steps in which at least one head moves is

$$\leq (t+1)^i \cdot m.$$

Altogether, for every run  $\rho$  of  $M$ , the total number of steps in which at least one head moves is

$$\leq \sum_{i=1}^r (t+1)^i \cdot m \leq (t+1)^{r+1} \cdot m. \tag{4.3}$$

Hence, we obtain that the total length of each run of  $M$  is

$$\leq k + k \cdot (t+1)^{r+1} \cdot m$$

(namely,  $M$  can pass through at most  $k$  configurations before moving a head for the first time, it can move a head for at most  $(t+1)^{r+1} \cdot m$  times, and between any two head movements, it can pass through at most  $k$  configurations).

Altogether, the proof of Lemma 4.8 is complete.  $\square$

## 5 List machines can simulate Turing machines

**Lemma 5.1 (Simulation Lemma).** *Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$ ,  $t \in \mathbb{N}$ , and let  $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$  be an  $(r, s, t)$ -bounded NTM with a total number of  $t+u$  tapes and with  $\{\square, \#\} \subseteq \Sigma$ . Then for every  $m, n \in \mathbb{N}$  there exists an  $(r(m \cdot (n+1)), t)$ -bounded NLM  $M_{m,n} = M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  with  $I = (\Sigma \setminus \{\square, \#\})^n$  and  $|C| \leq 2^{O(\ell(m \cdot (n+1)))}$ , where  $\ell(N)$  is an upper bound on the length of  $T$ 's runs on input words of length  $N$ , and*

$$|A| \leq 2^{d \cdot t^2 \cdot r(m \cdot (n+1)) \cdot s(m \cdot (n+1)) + 3t \cdot \log(m \cdot (n+1))}, \quad (5.1)$$

for some number  $d = d(u, |Q|, |\Sigma|)$  that does not depend on  $r, m, n, t$ , such that for all  $\bar{v} = (v_1, \dots, v_m) \in I^m$  we have

$$\Pr(M \text{ accepts } \bar{v}) = \Pr(T \text{ accepts } v_1 \# \dots v_m \#).$$

Furthermore, if  $T$  is deterministic, then  $M$  is deterministic, too.  $\dashv$

In Section 7 we will use the simulation lemma to transfer the lower bound results for list machines (obtained in Section 6) to lower bound results for Turing machines.

For proving Lemma 5.1, the following straightforward characterization of probabilities for Turing machines is very convenient.

**Definition 5.2 ( $C_T$  and  $\rho_T(w, \bar{c})$ ).** Let  $T$  be an NTM for which there exists a function  $\ell : \mathbb{N} \rightarrow \mathbb{N}$  such that every run of  $T$  on a length  $N$  input word has length at most  $\ell(N)$ . Let  $b := \max\{|\text{Next}_T(\gamma)| : \gamma \text{ is a configuration of } T\}$  be the maximum branching degree of  $T$  (note that  $b$  is finite since  $T$ 's transition relation is finite). Let  $b' := \text{lcm}\{1, \dots, b\}$  be the least common multiple of the numbers  $1, \dots, b$ , and let  $C_T := \{1, \dots, b'\}$ . For every  $N \in \mathbb{N}$ , every input word  $w \in \Sigma^*$  of length  $N$ , and every sequence  $\bar{c} = (c_1, \dots, c_{\ell(N)}) \in (C_T)^{\ell(N)}$ , we define  $\rho_T(w, \bar{c})$  to be the run  $(\rho_1, \dots, \rho_k)$  of  $T$  that is obtained by starting  $T$  with input  $w$  and by choosing in its  $i$ -th computation step the  $(c_i \bmod |\text{Next}_T(\rho_i)|)$ -th of the  $|\text{Next}_T(\rho_i)|$  possible next configurations.  $\dashv$

**Lemma 5.3.** *Let  $T$  be an NTM for which there exists a function  $\ell : \mathbb{N} \rightarrow \mathbb{N}$  such that every run of  $T$  on a length  $N$  input word has length at most  $\ell(N)$ , and let  $C_T$  be chosen according to Definition 5.2. Then we have for every run  $\rho$  of  $T$  on an input  $w$  of length  $N$  that*

$$(a) \quad \Pr(\rho) = \frac{|\{\bar{c} \in (C_T)^{\ell(N)} : \rho_T(w, \bar{c}) = \rho\}|}{|(C_T)^{\ell(N)}|}, \text{ and}$$

$$(b) \quad \Pr(T \text{ accepts } w) = \frac{|\{\bar{c} \in (C_T)^{\ell(N)} : \rho_T(w, \bar{c}) \text{ accepts}\}|}{|(C_T)^{\ell(N)}|}.$$

*Proof:* To prove (a), observe that for every run  $\rho = (\rho_1, \dots, \rho_k)$  of  $T$  on  $w$  we have  $k \leq \ell(N)$ , and

$$\begin{aligned} \frac{|\{\bar{c} \in C_T^{\ell(N)} : \rho_T(w, \bar{c}) = \rho\}|}{|C_T^{\ell(N)}|} &= \frac{1}{|C_T^{\ell(N)}|} \cdot \left( \prod_{i=1}^{k-1} \frac{|C_T|}{|\text{Next}_T(\rho_i)|} \right) \cdot |C_T|^{\ell(N) - (k-1)} \\ &= \prod_{i=1}^{k-1} \frac{1}{|\text{Next}_T(\rho_i)|} = \Pr(\rho). \end{aligned}$$

(b) follows directly from (a), since

$$\begin{aligned} \Pr(T \text{ accepts } w) &\stackrel{\text{def}}{=} \sum_{\rho : \rho \text{ is an accepting run of } T \text{ on } w} \Pr(\rho) \\ &\stackrel{(a)}{=} \sum_{\rho : \rho \text{ is an accepting run of } T \text{ on } w} \frac{|\{\bar{c} \in C_T^{\ell(N)} : \rho_T(w, \bar{c}) = \rho\}|}{|C_T^{\ell(N)}|} \\ &= \sum_{\substack{\bar{c} \in C_T^{\ell(N)} : \\ \rho_T(w, \bar{c}) \text{ accepts}}} \frac{1}{|C_T^{\ell(N)}|} \\ &= \frac{|\{\bar{c} \in C_T^{\ell(N)} : \rho_T(w, \bar{c}) \text{ accepts}\}|}{|C_T^{\ell(N)}|}. \quad \square \end{aligned}$$

**Outline of the proof of Lemma 5.1:**

For proving Lemma 5.1, let  $T$  be an NTM. We construct an NLM  $M$  that simulates  $T$ . The lists of  $M$  represent the external memory tapes of  $T$ . More precisely, the cells of the lists of  $M$  represent segments, or *blocks*, of the corresponding external memory tapes of  $T$  in such a way that the content of a block at any step of the computation can be reconstructed from the content of the cell representing it. The blocks evolve dynamically in a way that is described below.  $M$ 's set  $C$  of *nondeterministic choices* is defined as  $C := (C_T)^\ell$ , where  $C_T$  is chosen according to Definition 5.2 and  $\ell := \ell(m \cdot (n + 1))$  is an upper bound on  $T$ 's running time and tape length, obtained from Lemma 2.4. Each step of the list machine corresponds to the sequence of Turing machine steps that are performed by  $T$  while none of its external memory tape heads changes its direction or leaves its current tape block. Of course, the length  $\ell'$  of this sequence of  $T$ 's steps is bounded by  $T$ 's entire running time  $\ell$ . Thus, if  $c = (c_1, \dots, c_\ell) \in C = (C_T)^\ell$  is the nondeterministic choice used in  $M$ 's current step, the prefix of length  $\ell'$  of  $c$  tells us, which nondeterministic choices (in the sense of Definition 5.2)  $T$  makes throughout the corresponding sequence of  $\ell'$  steps. The *states* of  $M$  encode:

- The current state of the Turing machine  $T$ .
- The content and the head positions of the internal memory tapes  $t + 1, \dots, t + u$  of  $T$ .
- The head positions of the external memory tapes  $1, \dots, t$ .
- For each of the external memory tapes  $1, \dots, t$ , the boundaries of the block in which the head currently is.

Representing  $T$ 's current state and the content and head positions of the  $u$  internal memory tapes requires  $|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n + 1))^u$  states. The  $t$  head positions of the external memory tapes increase the number of states by a factor of  $\ell^t$ . The  $2t$  block boundaries increase the number of states by another factor of  $\ell^{2t}$ . So overall, the number of states is bounded by  $|Q| \cdot 2^{O(s(m \cdot (n+1)))} \cdot s(m \cdot (n + 1))^u \cdot \ell^{3t}$ . By Lemma 2.4, this yields the bound (5.1).

Initially, for an input word  $v_1\# \dots v_m\#$ , the first Turing machine tape is split into  $m$  blocks which contain the input segments  $v_i\#$  (for  $1 \leq i < m$ ), respectively,  $v_m\#\square^{\ell-(n+1)}$  (that is, the  $m$ -th input segment is padded by as many blank symbols as the Turing machine may enter throughout its computation). All other tapes just consist of one block which contains the blank string  $\square^\ell$ . The heads in the initial configuration of  $M$  are on the first cells of their lists. Now we start the simulation: For a particular nondeterministic choice  $c_1 = (c_{11}, c_{12}, \dots, c_{1\ell}) \in C = (C_T)^\ell$ , we start  $T$ 's run  $\rho_T(v_1\# \dots, c_{11}c_{12}c_{13} \dots)$ . As long as no head of the external memory tapes of  $T$  changes its direction or crosses the boundaries of its current block,  $M$  does not do anything. If a head on a tape  $i_0 \in \{1, \dots, t\}$  crosses the boundaries of its block, the head  $i_0$  of  $M$  moves to the next cell, and the previous cell is overwritten with sufficient information so that if it is visited again later, the content of the corresponding block of tape  $i_0$  of  $T$  can be reconstructed. The blocks on all other tapes are split behind the current head position (“behind” is defined relative to the current direction in which the head moves). A new cell is inserted into the lists behind the head, this cell represents the newly created tape block that is behind the head. The newly created block starting with the current head position is represented by the (old) cell on which the head still stands. The case that a head on a tape  $i_0 \in \{1, \dots, t\}$  changes its direction is treated similarly.

The simulation stops as soon as  $T$  has reached a final state; and  $M$  accepts if, and only if,  $T$  does. A close look at the possible runs of  $T$  and  $M$  shows that  $M$  has the same acceptance probabilities as  $T$ .

**Proof of Lemma 5.1:**

Let  $T = (Q, \Sigma, \Delta, q_0, F, F_{acc})$  be the given  $(r, s, t)$ -bounded nondeterministic Turing machine with  $t + u$  tapes, where the tapes  $1, \dots, t$  are the external memory tapes and tapes  $t + 1, \dots, t + u$  are the internal memory tapes. Let  $m, n \in \mathbb{N}$ . Recall that  $I = (\Sigma \setminus \{\square, \#\})^n$ . Let  $N = m \cdot (n + 1)$ . Every tuple  $\bar{v} = (v_1, \dots, v_m) \in I^m$  corresponds to an input string  $\tilde{v} := v_1\#v_2\# \dots v_m\#$  of length  $N$ . Let  $r := r(N)$  and  $s := s(N)$ .

By Lemma 2.4, there is a constant  $c_1 = c_1(u, |Q|, |\Sigma|)$ , which does *not* depend on  $r, m, n, t$ , such that every run of  $T$  on every input  $\bar{v}$ , for any  $\bar{v} \in I^m$ , has length at most

$$\ell(N) := N \cdot 2^{c_1 \cdot r \cdot (t+s)} \quad (5.2)$$

and throughout each such run, each of  $T$ 's external memory tapes  $1, \dots, t$  has length  $\leq \ell(N)$ . We let  $\ell := \ell(N)$ .

**Step 1:** *Definition of  $M$ 's set  $C$  of nondeterministic choices.*

$M$ 's set  $C$  of *nondeterministic choices* is chosen as  $C := (C_T)^\ell$ , where  $C_T$  is chosen according to Definition 5.2. ⊣

**Step 2:** *Definition of a superset  $\tilde{A}$  of  $M$ 's state set  $A$ .*

Let  $\hat{Q}$  be the set of potential configurations of tapes  $t+1, \dots, t+u$ , together with the current state of  $T$ , that is,

$$\hat{Q} := \left\{ (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}) \mid q \in Q, \text{ and for all } i \in \{1, 2, \dots, u\}, \right. \\ \left. p_{t+i} \in \{1, \dots, s\} \text{ and } w_{t+i} \in \Sigma^{\leq s} \right\}.$$

Then for a suitable constant  $c_2 = c_2(u, |Q|, |\Sigma|)$  we have

$$|\hat{Q}| \leq 2^{c_2 \cdot s}. \quad (5.3)$$

We let

$$\tilde{A} := \left\{ (\hat{q}, \bar{p}_1, \dots, \bar{p}_t) \mid \hat{q} \in \hat{Q}, \text{ and for each } j \in \{1, \dots, t\}, \right. \\ \bar{p}_j = (p_j^\parallel, p_j^\uparrow, p_j^\downarrow, \text{head-direction}_j), \text{ where} \\ p_j^\uparrow \in \{1, \dots, \ell\}, \text{head-direction}_j \in \{+1, -1\}, \text{ and} \\ \left. \text{either } p_j^\parallel = p_j^\downarrow = \ominus, \text{ or } p_j^\parallel, p_j^\downarrow \in \{1, \dots, \ell\} \text{ with } p_j^\parallel \leq p_j^\uparrow \leq p_j^\downarrow \right\}.$$

Here,  $\ominus$  is a symbol for indicating that  $p_j^\parallel$  and  $p_j^\downarrow$  are “undefined”, that is, that they cannot be interpreted as positions on one of the Turing machine's external memory tapes.

Later, at the end of Step 4, we will specify *which* particular subset of  $\tilde{A}$  will be designated as  $M$ 's state set  $A$ . With *any* choice of  $A$  as a subset of  $\tilde{A}$  we will have

$$|A| \leq |\tilde{A}| \leq |\hat{Q}| \cdot (\ell + 1)^{3 \cdot t} \cdot 2^t \leq 2^{c_2 \cdot s} \cdot (N \cdot 2^{c_1 \cdot r \cdot (t+s)} + 1)^{3 \cdot t} \cdot 2^t \leq 2^{d \cdot t^2 \cdot r \cdot s + 3 \cdot t \cdot \log N}$$

for a suitable constant  $d = d(u, |Q|, |\Sigma|)$ . This completes Step 2. ⊣

**Step 3:** *Definition of  $M$ 's initial state  $a_0$  and  $M$ 's sets  $B$  and  $B_{acc}$  of final states and accepting states, respectively.*

Let

$$\hat{q}_0 := (q_0, \underbrace{1, \dots, 1}_u, \underbrace{\square^s, \dots, \square^s}_u)$$

be the part of  $T$ 's *initial configuration* that describes the (start) state  $q_0$  of  $T$  and the head positions and initial (i.e., empty) content of the tapes  $t+1, \dots, t+u$  (that is, the tapes that represent internal memory).

Let

$$\bar{p}_1 := (p_1^\parallel, p_1^\uparrow, p_1^\downarrow, \text{head-direction}_1) := \begin{cases} (1, 1, n+1, +1) & \text{if } m \geq 2, \\ (1, 1, \ell, +1) & \text{otherwise,} \end{cases}$$

and for all  $i \in \{2, \dots, t\}$ ,

$$\bar{p}_i := (p_i^{\llbracket}, p_i^{\uparrow}, p_i^{\rrbracket}, \text{head-direction}_i) := (1, 1, \ell, +1).$$

As start state of the NLM  $M$  we choose

$$a_0 := (\hat{q}_0, \bar{p}_1, \bar{p}_2, \dots, \bar{p}_t).$$

As  $M$ 's sets of *final* and *accepting* states we choose  $B := \tilde{B} \cap A$  and  $B_{acc} := \tilde{B}_{acc} \cap A$ , respectively, with

$$\begin{aligned} \tilde{B} &:= \{(\hat{q}, \bar{p}_1, \bar{p}_2, \dots, \bar{p}_t) \in \tilde{A} \mid \hat{q} \text{ is of the form } (q, \bar{p}, \bar{y}) \in \hat{Q} \text{ for some } q \in F\}, \\ \tilde{B}_{acc} &:= \{(\hat{q}, \bar{p}_1, \bar{p}_2, \dots, \bar{p}_t) \in \tilde{A} \mid \hat{q} \text{ is of the form } (q, \bar{p}, \bar{y}) \in \hat{Q} \text{ for some } q \in F_{acc}\}. \end{aligned}$$

I.e., a state of  $M$  is *final* (resp., *accepting*) if, and only if, the associated state of the Turing machine  $T$  is. This completes Step 3.  $\dashv$

**Step 4:** Definition of  $M$ 's transition function  $\alpha : (A \setminus B) \times (\mathbf{A})^t \times C \rightarrow (A \times \text{Movement}^t)$ .

We let

$$\begin{aligned} \text{Conf}_T &:= \{ (q, p_1, \dots, p_{t+u}, w_1, \dots, w_{t+u}) \mid q \in Q, \\ &\quad \text{for all } j \in \{1, \dots, t+u\}: p_j \in \mathbb{N}, \\ &\quad \text{for all } j \in \{1, \dots, t\}: w_j \in \{\otimes\}^* \Sigma^* \{\otimes\}^* \text{ with } w_{j,p_j} \in \Sigma, \\ &\quad \text{for all } j \in \{1, \dots, u\}: w_{t+j} \in \Sigma^* \}, \end{aligned}$$

where  $\otimes$  is a symbol *not* in  $\Sigma$ , and  $w_{j,p_j}$  denotes the  $p_j$ -th letter in the string  $w_j$ . The symbol  $\otimes$  is used as a *wildcard* symbol that may be interpreted by any symbol in  $\Sigma$ . An element in  $\text{Conf}_T$  gives (potentially) incomplete information on a configuration of  $T$ , where the contents of tapes  $1, \dots, t$  might be described only in some part (namely, in the part containing no  $\otimes$ -symbols).

We let  $\tilde{A} := I \cup C \cup \tilde{A} \cup \{\langle, \rangle\}$ . In what follows, we inductively fix for every  $i \geq 0$ ,

- a set  $A_i \subseteq \tilde{A}$ ,
- a set  $K_i \subseteq (\tilde{A} \setminus \tilde{B}) \times (\tilde{A}^*)^t$ ,
- a set  $L_i \subseteq \tilde{A}^*$  defined by

$$L_i := \{ \langle a \langle y_1 \rangle \dots \langle y_t \rangle c \mid (a, y_1, \dots, y_t) \in K_i \text{ and } c \in C \}, \quad (5.4)$$

- a function

$$\text{config}_i : K_i \rightarrow \text{Conf}_T \cup \{\perp\},$$

(Intended meaning: When the NLM  $M$  is in a situation  $\kappa \in K_i$ , then  $\text{config}_i(\kappa)$  is the Turing machine's configuration at the beginning of  $M$ 's current step. If  $\text{config}_i(\kappa) = \perp$ , then  $\kappa$  does not represent a configuration of the Turing machine.)

- the transition function  $\alpha$  of  $M$ , restricted to  $K_i$ , that is,

$$\alpha_{|K_i} : K_i \times C \rightarrow \tilde{A} \times \text{Movement}^t,$$

- for every tape  $j \in \{1, \dots, t\}$ , a function

$$\begin{aligned} \text{tape-config}_{j,i} : L_i &\rightarrow \{ (w, p^{\llbracket}, p^{\rrbracket}) \mid \text{either } 1 \leq p^{\llbracket} \leq p^{\rrbracket} \leq \ell \text{ and } w \in \{\otimes\}^{p^{\llbracket}-1} \Sigma^{p^{\rrbracket}-p^{\llbracket}+1} \{\otimes\}^{\ell-p^{\rrbracket}}, \\ &\quad \text{or } p^{\llbracket} > p^{\rrbracket} \text{ and } w = \varepsilon \}. \end{aligned}$$

(Intended meaning: When the NLM  $M$  is in a situation  $\kappa = (a, y_1, \dots, y_t) \in K_i$  and nondeterministically chooses  $c \in C$  for its current transition, then

$$\text{tape-config}_{j,i}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$$

gives information on the inscription from tape cell  $p^{\llbracket}$  up to tape cell  $p^{\rrbracket}$  of the  $j$ -th tape of the Turing machine's configuration at the end of  $M$ 's current step.)

*Induction base* ( $i = 0$ ): We start with  $M$ 's start state  $a_0$  and choose

$$A_0 := \{a_0\}.$$

If  $a_0$  is *final*, then we let  $K_0 := \emptyset$  and  $A := A_0$ . This then gives us an NLM  $M$  which accepts its input without performing a single step. This is fine, since  $a_0$  is final if, and only if, the Turing machine  $T$ 's start state  $q_0$  is final. That is,  $T$  accepts its input without performing a single step. For the case that  $a_0$  is *not* final, we let

$$K_0 := \{ (a_0, y_1, \dots, y_t) \mid y_1 = \langle v \rangle \text{ for some } v \in I, \text{ and } y_2 = \dots = y_t = \langle \rangle \}.$$

The set  $L_0$  is defined as in (5.4).

The functions  $config_0$ ,  $\alpha_{|K_0}$  and  $tape-config_{j,0}$  are defined as follows. Let

$$\kappa = (a_0, y_1, \dots, y_t) \in K_0,$$

where  $y_1 = \langle v \rangle$  for some  $v \in I$ . That is,  $\kappa$  describes the situation of the list machine  $M$ , where  $M$  is in the start state  $a_0$ , the entry of the current list cell in the input list is  $y_1 = \langle v \rangle$ , and for each  $j \in \{2, \dots, t\}$ , the entry of the current list cell in list  $j$  is  $y_j = \langle \rangle$ . Recall from step 3 that

$$a_0 = (\hat{q}_0, \bar{p}_1, \dots, \bar{p}_t),$$

where

$$\hat{q}_0 := (q_0, \underbrace{1, \dots, 1}_u, \underbrace{\square^s, \dots, \square^s}_u)$$

is the part of  $T$ 's *initial configuration* that describes the (start) state  $q_0$  of  $T$  and the head positions and initial (i.e., empty) content of the tapes  $t+1, \dots, t+u$ , and for all  $j \in \{1, \dots, t\}$ ,

$$\bar{p}_j = (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}, head-direction_j) = \begin{cases} (1, 1, n+1, +1) & \text{if } j = 1 \text{ and } m \geq 2, \\ (1, 1, \ell, +1) & \text{otherwise.} \end{cases}$$

That is,  $\kappa$  corresponds to the (partial) configuration

$$config_0(\kappa) := (q_0, \overbrace{1, \dots, 1}^{t+u}, v\#\otimes^{\ell-(n+1)}, \underbrace{\square^\ell, \dots, \square^\ell}_{t-1}, \underbrace{\square^s, \dots, \square^s}_u).$$

of  $T$  at the beginning of the computation on a string of the form  $v\#\dots$ . I.e., the situation where  $T$  is in the start state  $q_0$ , the head of each tape is placed on the first tape cell, the input tape holds the input string (which starts with  $v\#$ , and we don't care about the remaining  $\ell - (n+1)$  symbols on the input tape), and all other tapes contain only blank symbols (see Figure 2).

For the definition of  $\alpha_{|K_0}$  and  $tape-config_{j,0}$ , let  $c = (c_1, c_2, \dots, c_\ell) \in C = (C_T)^\ell$  be an arbitrary element from  $M$ 's set  $C$  of nondeterministic choices. Then,  $\alpha_{|K_0}(\kappa, c)$  and  $tape-config_{j,0}(\langle a_0 \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$  are defined as follows: Let us start the Turing machine  $T$  with a configuration  $\gamma_1$  that *fits* to  $config_0(\kappa)$ , i.e., that can be obtained from  $config_0(\kappa)$  by replacing each occurrence of the wildcard symbol  $\otimes$  by an arbitrary symbol in  $\Sigma$ . Let  $\gamma_1, \gamma_2, \gamma_3, \dots$  be the successive configurations of  $T$  when started in  $\gamma_1$  and using the nondeterministic choices  $c_1, c_2, c_3, \dots$  (in the sense of Definition 5.2). That is, for all  $v \geq 1$ ,  $\gamma_{v+1}$  is the  $(c_v \bmod |\text{Next}_T(\gamma_v)|)$ -th of the  $|\text{Next}_T(\gamma_v)|$  possible next configurations of  $\gamma_v$ .

Using this notation, and letting  $(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket}) := (p_j^{\llbracket}, p_j^{\rrbracket})$  for all  $j \in \{1, \dots, t\}$ , the definition of

$$\alpha_{|K_0}(\kappa, c) \quad \text{and} \quad tape-config_{j,0}(\langle a_0 \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$$

can be taken verbatim from the definition of

$$\alpha_{|K_{i+1}}(\kappa, c) \quad \text{and} \quad tape-config_{j,i+1}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$$

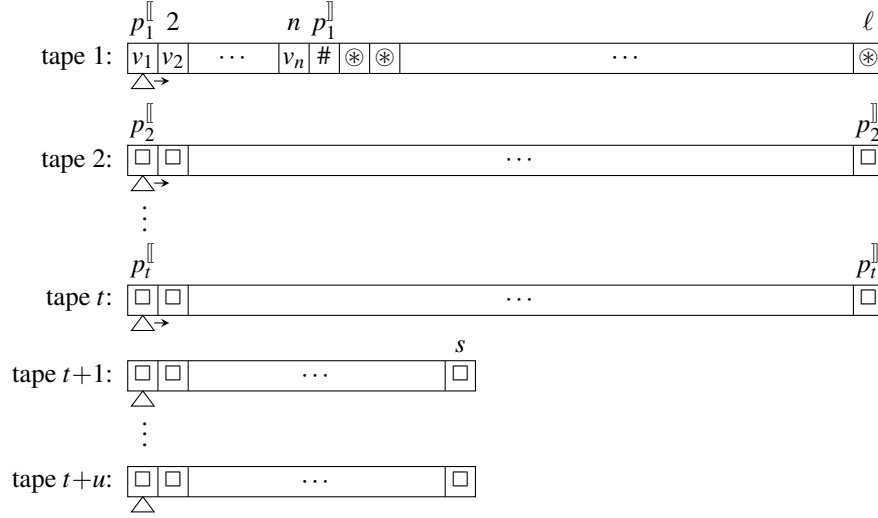


Figure 2: The (partial) configuration of  $T$  described by  $config_0(\kappa)$ . We assume that  $v = v_1 v_2 \cdots v_n$  and  $m \geq 2$ . The picture also shows the left and right block boundaries  $p_j^{\llbracket}, p_j^{\rrbracket}$  as well as the head direction for each tape  $j \in \{1, \dots, t\}$ .

given in the induction step below (except that  $a$  has to be replaced by  $a_0$ ). This completes the induction base.

*Induction step ( $i \rightarrow i+1$ ):* We let

$$A_{i+1} := \left\{ b \in \tilde{A} \mid \text{there are } \kappa \in K_i, c \in C, \text{ and } (e_1, \dots, e_t) \in \text{Movement}^t \right. \\ \left. \text{such that } \alpha_{|K_i}(\kappa, c) = (b, e_1, \dots, e_t) \right\},$$

and

$$K_{i+1} := \left\{ (a, y_1, \dots, y_t) \mid a \in A_{i+1} \setminus \tilde{B}, y_1 \in \{\langle v \rangle \mid v \in I\} \cup \bigcup_{i' \leq i} L_{i'}^t, \text{ and} \right. \\ \left. \text{for all } j \in \{2, \dots, t\} \text{ we have } y_j \in \{\langle \rangle\} \cup \bigcup_{i' \leq i} L_{i'}^t \right\}.$$

The set  $L_{i+1}$  is defined via equation (5.4).

The functions  $config_{i+1}$ ,  $\alpha_{|K_{i+1}}$  and  $tape\text{-}config_{j,i+1}$  are defined as follows. Let  $\kappa = (a, y_1, \dots, y_t) \in K_{i+1}$ , and let  $c \in C$ . That is,  $\kappa$  describes the situation of the list machine  $M$ , where  $M$  is in state  $a$ , and the entry of the current list cell in the  $j$ -th list is  $y_j$ . We assume that

$$a = (\hat{q}, \bar{p}_1, \dots, \bar{p}_t),$$

where

$$\hat{q} = (q, p_{t+1}, \dots, p_{t+u}, w_{t+1}, \dots, w_{t+u}),$$

and

$$\bar{p}_j = (p_j^{\llbracket}, p_j^{\uparrow}, p_j^{\rrbracket}, \text{head-direction}_j)$$

for all  $j \in \{1, \dots, t\}$ . Basically,  $M$ 's current state  $a$  represents the (partial) configuration of  $T$ , where

- $T$  is in state  $q$ ,
- tape  $t+j$  (for  $1 \leq j \leq u$ ) contains the string  $w_{t+j}$ , and its head is placed on the  $p_{t+j}$ -th tape cell,
- the head of tape  $j$  (for  $1 \leq j \leq t$ ) is on position  $p_j^{\uparrow}$ , in the block whose leftmost tape cell is at position  $p_j^{\llbracket}$  and whose rightmost tape cell is at position  $p_j^{\rrbracket}$ , and

- the current block on tape  $j$  (for  $1 \leq j \leq t$ ) has been “entered” from the left if  $head-direction_j = +1$ , or from the right if  $head-direction_j = -1$ .

See Figure 3 for an illustration. We say “basically”, because  $p_j^{\llbracket}$  and  $p_j^{\rrbracket}$  might be undefined (i.e.,  $p_j^{\llbracket} =$

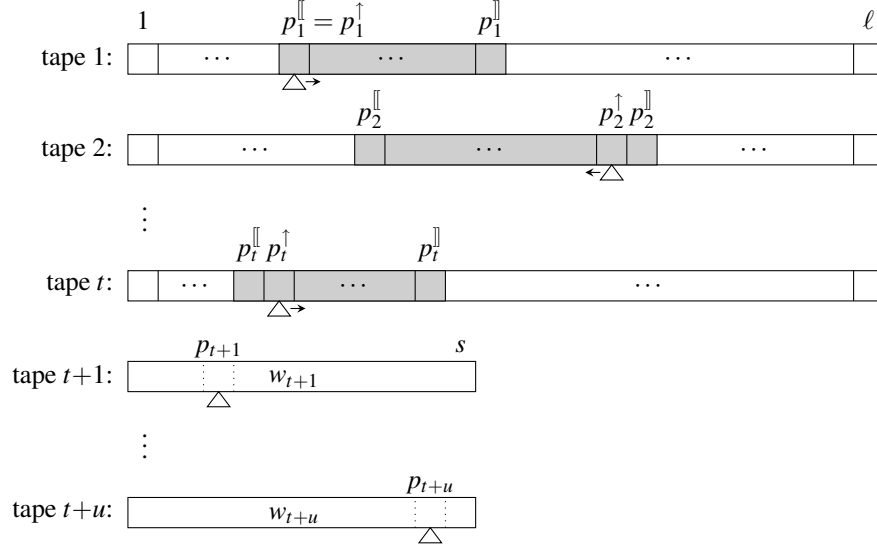


Figure 3: A possible (partial) configuration of  $T$  represented by  $a$ .

$p_j^{\rrbracket} = \ominus$ ) for some  $j \in \{1, \dots, t\}$ . Intuitively, this occurs if the current block on the  $j$ -th tape of  $T$  is a block that has just been “entered”. (In particular, such a situation is created by case 1 below.)

In the remainder of step 4, we first use, for each  $j \in \{1, \dots, t\}$ , the list entry  $y_j$  to adjust the boundaries of the current block on tape  $j$  (in case they are undefined), and to extract the inscriptions of the tape cells within this block. Based on this information, we then define  $config_{i+1}(\kappa)$ ,  $\alpha_{|K_{i+1}}(\kappa, c)$ , and  $tape-config_{j,i+1}(\langle a \rangle_{y_1} \dots \langle y_t \rangle c)$ .

Let  $j \in \{1, \dots, t\}$ . Then the boundaries of the block on tape  $j$ , and the inscriptions of the tape cells within this block are obtained from the list entry  $y_j$  as follows. We distinguish between the following three cases:

*Case I:*  $y_j \in L_{i'}$  for some  $i' \leq i$ . In this case, the list entry  $y_j$  intuitively corresponds to a block on the  $j$ -th tape of  $T$  that has already been visited before during the simulation of  $T$  by  $M$ , and

$$tape-config_{j,i'}(y_j) = (w'_j, p'^{\llbracket}_j, p'^{\rrbracket}_j)$$

gives us information on the inscriptions of the tape cells within that block, and the corresponding left and right boundary,  $p'^{\llbracket}_j$  and  $p'^{\rrbracket}_j$ .

We then set the boundaries  $(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket})$  of the current block on tape  $j$  to

$$(\hat{p}_j^{\llbracket}, \hat{p}_j^{\rrbracket}) := \begin{cases} (p_j^{\uparrow}, p'^{\rrbracket}_j) & \text{if } p_j^{\llbracket} = p_j^{\rrbracket} = \ominus \text{ and } head-direction_j = +1, \\ (p'^{\llbracket}_j, p_j^{\uparrow}) & \text{if } p_j^{\llbracket} = p_j^{\rrbracket} = \ominus \text{ and } head-direction_j = -1, \\ (p_j^{\llbracket}, p_j^{\rrbracket}) & \text{otherwise.} \end{cases}$$

So, intuitively, if the boundaries of the block on tape  $j$  are undefined, and the head of tape  $j$  “looks” to the right, then we adjust the boundaries such that the current block on tape  $j$  begins directly at the current head position  $p_j^{\uparrow}$  and ends at the last position  $p_j^{\rrbracket}$  at which  $w'_j$  contains a non-wildcard symbol; the case that the head of tape  $j$  “looks” to the left is analogous.

We also let

$$w_j := w'_j$$

(which is well-defined, because  $\text{tape-config}_{j,i'}$  and  $\text{tape-config}_{j,i''}$  operate identically on all elements in  $L_{i'} \cap L_{i''}$ , for all  $i', i'' \leq i$ ).

*Case II:  $j = 1$  and  $y_j \notin \bigcup_{i' \leq i} L_{i'}$ .* In this case,  $y_j$  is the content of a list cell in  $M$ 's input list that has not been visited before. Thus there is some  $v \in I$  such that  $y_j = \langle v \rangle$ . Note also that  $\text{head-direction}_j = +1$ .

First assume that  $v$  is *not* the last item in the input list. That is, there is some  $\mu \in \{1, \dots, m-1\}$  such that  $(\mu-1) \cdot (n+1) < p_j^\uparrow \leq \mu \cdot (n+1)$ . Then we set the boundaries  $(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket)$  of the block on tape  $j$  to

$$(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket) := (p_j^\uparrow, \mu \cdot (n+1)),$$

and let

$$w_j := \otimes^{(\mu-1) \cdot (n+1)} v \# \otimes^{\ell - \mu \cdot (n+1)}.$$

Next assume that  $v$  is the last item in the input list, i.e.,  $p_j^\uparrow > (m-1) \cdot (n+1)$ . Then we set the boundaries  $(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket)$  of the block on tape  $j$  to

$$(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket) := (p_j^\uparrow, \ell),$$

and let

$$w_j := \otimes^{(m-1) \cdot (n+1)} v \# \square^{\ell - m \cdot (n+1)}.$$

*Case III:  $j \in \{2, \dots, t\}$  and  $y_j \notin \bigcup_{i' \leq i} L_{i'}$ .* In this case,  $y_j$  is the content of a list cell on the  $j$ -th list of  $M$  that has not been visited before. Thus,  $y_j = \langle \rangle$  and  $\text{head-direction}_j = +1$ . We then set the boundaries  $(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket)$  of the current block on tape  $j$  to

$$(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket) := (p_j^\uparrow, \ell),$$

and define

$$w_j := \otimes^{\hat{p}_j^\llbracket - 1} \square^{\ell - (\hat{p}_j^\llbracket - 1)}.$$

This completes case III.

Now, for each  $j \in \{1, \dots, t\}$  we have the boundaries  $(\hat{p}_j^\llbracket, \hat{p}_j^\rrbracket)$  of the current block on tape  $j$ , and the string  $w_j$  on tape  $j$  (which may have wildcard symbols only at positions outside the current block).

Before we continue with the main part of the definition of  $\text{config}_{i+1}$ ,  $\alpha_{|K_{i+1}}$  and  $\text{tape-config}_{j,i+1}$ , let us deal with the following special case. Namely, observe that we might have  $w_{j_0} = \varepsilon$  for some  $j_0 \in \{1, \dots, t\}$ . In this case,  $y_{j_0}$  intuitively corresponds to an ‘‘empty’’ block on the  $j_0$ -th tape of  $T$ . Such an empty block is the result of splitting a larger block into two pieces: one empty piece, and one piece that contains the original block. (In particular, such a block can be created in case 1 and case 2 below.) To reach a list cell that corresponds to the next non-empty block on tape  $j_0$ , we move the head of list  $j_0$  to the next list cell. More precisely, if  $w_{j_0} = \varepsilon$  for some  $j_0 \in \{1, \dots, t\}$ , we define

$$\alpha_{|K_{i+1}}(\kappa, c) := (a, e_1, \dots, e_t),$$

where for all  $j \in \{1, \dots, t\}$ ,

$$e_j := \begin{cases} (\text{head-direction}_j, \text{true}) & \text{if } w_j = \varepsilon \\ (\text{head-direction}_j, \text{false}) & \text{otherwise.} \end{cases}$$

We also let

$$\text{config}_{i+1}(\kappa) := \perp \quad \text{and} \quad \text{tape-config}_{j,i+1}(\langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle) := (\varepsilon, 2, 1).$$

We are now ready to define  $config_{i+1}(\kappa)$ ,  $\alpha_{\kappa_{i+1}}(\kappa, c)$ , and  $tape-config_{j,i+1}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$ , where we can assume  $w_j \neq \varepsilon$  for all  $j \in \{1, \dots, t\}$ . First, we let

$$config_{i+1}(\kappa) := (q, p_1^\uparrow, \dots, p_t^\uparrow, p_{t+1}, \dots, p_{t+u}, w_1, \dots, w_t, w_{t+1}, \dots, w_{t+u})$$

be the (partial) configuration of  $T$ , where  $T$  is in state  $q$ , where tape  $j$  contains the string  $w_j$  (which, for  $j \in \{1, \dots, t\}$ , might have wildcard symbols outside the current block), and where the head of tape  $j$  is on position  $p_j^\uparrow$  if  $j \in \{1, \dots, t\}$ , and on position  $p_j$  if  $j \in \{t+1, \dots, t+u\}$ . This is illustrated in Figure 4.

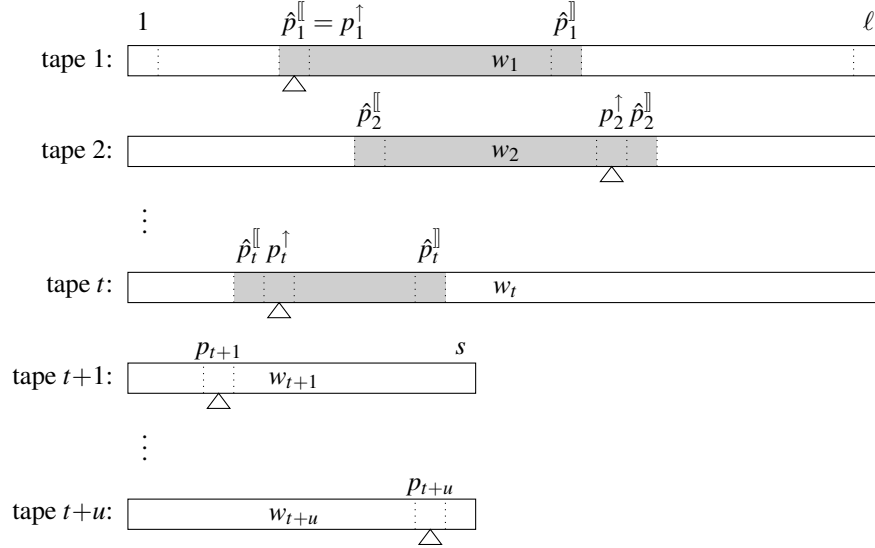


Figure 4: A possible (partial) configuration of  $T$  represented by  $config_{i+1}(\kappa)$ . The picture also shows the block boundaries of the current blocks on each of the first  $t$  tapes.

For the definition of  $\alpha_{\kappa_{i+1}}(\kappa, c)$  and  $tape-config_{j,i+1}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle)$ , we then consider the following: Let us start the Turing machine  $T$  with a configuration  $\gamma_1$  that fits to  $config_{i+1}(\kappa)$ , i.e., that can be obtained from  $config_{i+1}(\kappa)$  by replacing each occurrence of the wildcard symbol  $\otimes$  by an arbitrary symbol in  $\Sigma$ . Assuming

$$c = (c_1, c_2, \dots, c_\ell) \in C = (C_T)^\ell,$$

we let  $\gamma_1, \gamma_2, \gamma_3, \dots$  be the successive configurations of  $T$  when started in  $\gamma_1$  and using the nondeterministic choices  $c_1, c_2, c_3, \dots$  (in the sense of Definition 5.2). I.e., for all  $v \geq 1$ ,  $\gamma_{v+1}$  is the  $(c_v \bmod |\text{Next}_T(\gamma_v)|)$ -th of the  $|\text{Next}_T(\gamma_v)|$  possible next configurations of  $\gamma_v$ .

Then, there is a minimal  $v > 1$  for which there exists a  $j_0 \in \{1, \dots, t, \perp\}$  such that throughout the run  $\gamma_1 \dots \gamma_{v-1}$ ,

- (1) none of the heads  $1, \dots, t$  changes its direction, and
- (2) none of the heads  $j \in \{1, \dots, t\}$  crosses a border  $\hat{p}_j^{\llbracket}$  or  $\hat{p}_j^{\rrbracket}$ ,

and one of the following cases applies:

**Case 1:**  $j_0 \neq \perp$ , and in the transition from  $\gamma_{v-1}$  to  $\gamma_v$ , head  $j_0$  crosses one of the borders  $\hat{p}_{j_0}^{\llbracket}$  or  $\hat{p}_{j_0}^{\rrbracket}$ . That is, in  $\gamma_v$ , the  $j_0$ -th head is either at position  $\hat{p}_{j_0}^{\llbracket} - 1$  or at position  $\hat{p}_{j_0}^{\llbracket} + 1$ .

(And none of the heads  $j \in \{1, \dots, t\} \setminus \{j_0\}$  crosses a border or changes its direction.<sup>1</sup>)

**Case 2:**  $j_0 \neq \perp$ , and in the transition from  $\gamma_{v-1}$  to  $\gamma_v$ , head  $j_0$  changes its direction, but does not cross one of the borders  $\hat{p}_{j_0}^{\llbracket}$  or  $\hat{p}_{j_0}^{\rrbracket}$ .

(And none of the heads  $j \in \{1, \dots, t\} \setminus \{j_0\}$  crosses a border or changes its direction.)

<sup>1</sup>Recall that w.l.o.g. we assume that the Turing machine is normalized, cf. Definition 2.1.

**Case 3:**  $\gamma_v$  is *final* and none of the cases 1 and 2 apply. Then we let  $j_0 := \perp$ .

In the following, we assume that

$$\gamma_v = (q'', p_1'', \dots, p_{t+u}'', w_1'', \dots, w_{t+u}'').$$

**ad Case 1:** In this case we have  $j_0 \neq \perp$ , and head  $j_0$  crosses one of the borders  $\hat{p}_{j_0}^{\parallel}$  or  $\hat{p}_{j_0}'^{\parallel}$  in the transition from  $\gamma_{v-1}$  to  $\gamma_v$  (that is,  $p_{j_0}''$  is either  $\hat{p}_{j_0}^{\parallel} + 1$  or  $\hat{p}_{j_0}'^{\parallel} - 1$ ).

Intuitively, the head of tape  $j_0$  has just “left” the current block (represented by the current list cell on the  $j_0$ -th list of  $M$ ), and moved to the next block (represented by the next list cell on the  $j_0$ -th list of  $M$ ) whose boundaries are not yet known. (Figure 5 illustrates the situation where the current block has been left through the right boundary.) We represent the left boundary of the new block, the new head position,

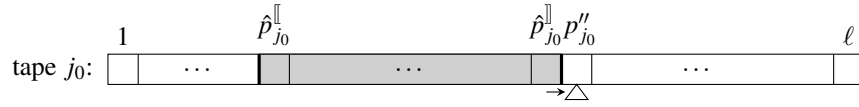


Figure 5: The situation directly after head  $j_0$  crossed the border  $\hat{p}_{j_0}'^{\parallel}$  of the current block on tape  $j_0$ .

the right boundary of the new block, and the direction of the head of tape  $j_0$  by the tuple

$$\bar{p}_{j_0}'' := (\ominus, p_{j_0}'', \ominus, head-direction_{j_0}''), \quad \text{where } head-direction_{j_0}'' := \begin{cases} +1, & \text{if } p_{j_0}'' = \hat{p}_{j_0}^{\parallel} + 1 \\ -1, & \text{if } p_{j_0}'' = \hat{p}_{j_0}'^{\parallel} - 1. \end{cases}$$

On all tapes  $j \in \{1, \dots, t\} \setminus \{j_0\}$ , we know (by the choice of  $v$  and  $j_0$ ) that throughout  $T$ 's computation  $\gamma_1, \dots, \gamma_v$ , head  $j$  neither changes its direction nor crosses one of the borders of the current block on tape  $j$  (which is represented by the current list cell on the  $j$ -th list of  $M$ ). We then “split” the current block on tape  $j$  directly behind the current head position. The part “in front of” (and including) the current head position then becomes the new current block on tape  $j$ , which will be represented by the current list cell. (See Figure 6 for an illustration.) We represent the left boundary of the new current

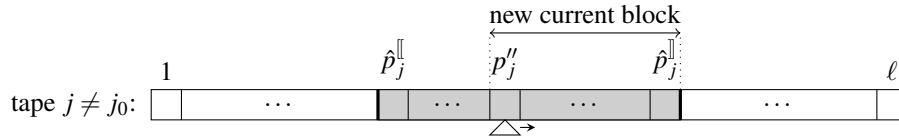


Figure 6: The situation on tape  $j \neq j_0$ , where the head of tape  $j$  “looks” to the right.

block, the new head position, the right boundary of the new current block, and the direction of the head of tape  $j$  by

$$\bar{p}_j'' := \begin{cases} (p_j'', p_j'', \hat{p}_j^{\parallel}, head-direction_j), & \text{if } head-direction_j = +1 \\ (\hat{p}_j^{\parallel}, p_j'', p_j'', head-direction_j), & \text{if } head-direction_j = -1. \end{cases}$$

The part “behind” the current head position will be represented by a new list cell that is inserted “behind” the current list cell.

We capture this in  $M$ 's transition function  $\alpha$  by letting

$$\alpha_{\kappa_{i+1}}(\kappa, c) := (b, e_1'', \dots, e_t'') \quad \text{with} \quad b := (\hat{q}'', \bar{p}_1'', \dots, \bar{p}_t''),$$

where

$$\hat{q}'' := (q'', p_{t+1}'', \dots, p_{t+u}'', w_{t+1}'', \dots, w_{t+u}'')$$

contains the state, and the configurations of tape  $t + 1$  to tape  $t + u$  from  $\gamma_v$ , and for all  $j \in \{1, \dots, t\}$ ,

$$e_j'' := \begin{cases} (\text{head-direction}_{j_0}'', \text{true}), & \text{if } j = j_0 \\ (\text{head-direction}_j'', \text{false}), & \text{if } j \neq j_0. \end{cases}$$

This ensures that on list  $j_0$ , the list machine  $M$  moves to the next list cell and overwrites the ‘‘old’’ one with the string  $\langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle$ , and on every other list  $j \neq j_0$ ,  $M$  inserts a new list cell behind the current one and writes the string  $\langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle$  into the new list cell.

Note that the string  $y' := \langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle$  contains enough information to reconstruct what has been written during the computation  $\gamma_1, \dots, \gamma_v$ . Namely,  $y'$  contains enough information to reconstruct the string

$$\tilde{w}_{j_0} := w_{j_0, \tilde{p}_{j_0}''}'' \cdots w_{j_0, \tilde{p}_{j_0}''}''$$

that is written on tape  $j_0$  in the tape cells from position  $\tilde{p}_{j_0}''$  to  $\tilde{p}_{j_0}''$ , where

$$(\tilde{p}_{j_0}'', \tilde{p}_{j_0}'' ) := (\hat{p}_{j_0}'', \hat{p}_{j_0}'' ).$$

And for  $j \in \{1, \dots, t\} \setminus \{j_0\}$ , the string  $y'$  contains enough information to reconstruct the string

$$\tilde{w}_j := w_{j, \tilde{p}_j}'' \cdots w_{j, \tilde{p}_j}''$$

that is written on tape  $j$  in the tape cells from position  $\tilde{p}_j''$  to  $\tilde{p}_j''$ , where

$$(\tilde{p}_j'', \tilde{p}_j'' ) := \begin{cases} (\hat{p}_j'', p_j'' - 1) & \text{if } \text{head-direction}_j = +1 \\ (p_j'' + 1, \hat{p}_j'') & \text{if } \text{head-direction}_j = -1. \end{cases}$$

To extract this information for each  $j \in \{1, \dots, t\}$ , we use the function  $\text{tape-config}_{j,i+1}$ , which is defined on  $y' = \langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle$  by

$$\text{tape-config}_{j,i+1}(\langle a \langle y_1 \rangle \cdots \langle y_t \rangle c \rangle) := \begin{cases} (\otimes^{\tilde{p}_j''-1} \tilde{w}_j \otimes^{\ell-\tilde{p}_j''+1}, \tilde{p}_j'', \tilde{p}_j'') & \text{if } \tilde{p}_j'' \leq \tilde{p}_j'', \\ (\varepsilon, \tilde{p}_j'', \tilde{p}_j'') & \text{otherwise.} \end{cases}$$

Notice that the situation  $\tilde{p}_j'' > \tilde{p}_j''$  can occur only if  $j \neq j_0$ , and the head of tape  $j$  does not move during the computation  $\gamma_1, \dots, \gamma_v$  of  $T$ , so that we split the current block on tape  $j$  into two parts, one of which is ‘‘empty’’, and the other one corresponds to the complete original block.

**ad Case 2:** In this case we have  $j_0 \neq \perp$ , and head  $j_0$  changes its direction, but does not cross one of the borders  $\hat{p}_{j_0}''$  or  $\hat{p}_{j_0}''$ . We only consider the case where the direction of head  $j_0$  changes from  $+1$  to  $-1$  (the other case is symmetric).

Thus, in configuration  $\gamma_{v-1}$ , the head of tape  $j_0$  is on position  $p_{j_0}'' + 1$ , and in configuration  $\gamma_v$ , the head of tape  $j_0$  is on position  $p_{j_0}''$ . We then ‘‘split’’ the current block on tape  $j_0$  (which is represented by the current list cell in the  $j_0$ -th list of  $M$ ) directly behind position  $p_{j_0}'' + 1$ . The part ‘‘in front of’’ (and including) position  $p_{j_0}'' + 1$  then becomes the new current block on tape  $j_0$  (which will be represented by the current list cell in the  $j_0$ -th list of  $M$ ). This is illustrated in Figure 7. We represent the left boundary of the new current block, the new head position, the right boundary of the new current block, and the direction of the head of tape  $j_0$  by the tuple

$$\bar{p}_{j_0}'' := (\hat{p}_{j_0}'', p_{j_0}'', p_{j_0}'' + 1, -1).$$

The part ‘‘behind’’ position  $p_{j_0}'' + 1$  will be represented by a new list cell that is inserted ‘‘behind’’ the current list cell.

On all tapes  $j \in \{1, \dots, t\} \setminus \{j_0\}$ , we know (by the choice of  $v$  and  $j_0$ ) that throughout  $T$ 's computation  $\gamma_1, \dots, \gamma_v$ , head  $j$  neither changes its direction nor crosses one of the borders of the current block on

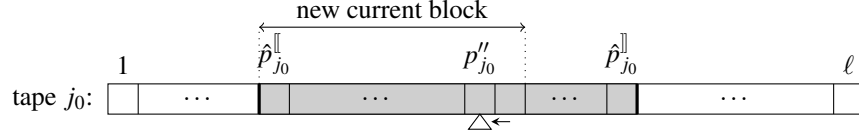


Figure 7: The situation directly after head  $j_0$  changed its direction from  $+1$  (right) to  $-1$  (left).

tape  $j$ . We treat these tapes in precisely the same way as in case 1 above. In particular, we “split” the current block on tape  $j$  (which is represented by the current list cell in the  $j$ -th list of  $M$ ) directly behind the current head position. The part “in front of” (and including) the current head position then becomes the new current block on tape  $j$ , which will be represented by the current list cell. (See Figure 6 for an illustration.) We let

$$\bar{p}_j := \begin{cases} (p''_j, p''_j, \hat{p}_j, \text{head-direction}_j), & \text{if } \text{head-direction}_j = +1 \\ (\hat{p}_j, p''_j, p''_j, \text{head-direction}_j), & \text{if } \text{head-direction}_j = -1. \end{cases}$$

We capture this in  $M$ 's transition function  $\alpha$  by letting

$$\alpha_{\kappa_{i+1}}(\kappa, c) := (b, e''_1, \dots, e''_t) \quad \text{with} \quad b := (\hat{q}'', \bar{p}_1'', \dots, \bar{p}_t''),$$

where

$$\hat{q}'' := (q'', p''_{t+1}, \dots, p''_{t+u}, w''_{t+1}, \dots, w''_{t+u})$$

is defined in the same way as in case 1 above, and for all  $j \in \{1, \dots, t\}$ ,

$$e''_j := \begin{cases} (-1, \text{false}), & \text{if } j = j_0 \\ (\text{head-direction}_j, \text{false}), & \text{if } j \neq j_0. \end{cases}$$

This ensures that on every list,  $M$  inserts a new list cell behind the current one and writes the string  $\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle$  into the new list cell.

Note that the string  $y' := \langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle$  contains enough information to reconstruct the string

$$\tilde{w}_{j_0} := w''_{j_0, \bar{p}_{j_0}''} \dots w''_{j_0, \bar{p}_{j_0}''}$$

that is written on tape  $j_0$  in the tape cells from position  $\bar{p}_{j_0}''$  to  $\bar{p}_{j_0}''$ , where

$$(\bar{p}_{j_0}'', \bar{p}_{j_0}'') := (p''_{j_0} + 2, \hat{p}_{j_0}'').$$

Moreover, for each  $j \in \{1, \dots, t\} \setminus \{j_0\}$ , the string  $y'$  contains enough information to reconstruct the string

$$\tilde{w}_j := w''_{j, \bar{p}_j''} \dots w''_{j, \bar{p}_j''}$$

that is written on tape  $j$  in the tape cells from position  $\bar{p}_j''$  to  $\bar{p}_j''$ , where

$$(\bar{p}_j'', \bar{p}_j'') := \begin{cases} (\hat{p}_j'', p''_j - 1) & \text{if } \text{head-direction}_j = +1 \\ (p''_j + 1, \hat{p}_j'') & \text{if } \text{head-direction}_j = -1. \end{cases}$$

To extract this information for each  $j \in \{1, \dots, t\}$ , we use the function  $\text{tape-config}_{j,i+1}$ , which is defined on  $y' = \langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle$  in precisely the same way as in case 1, namely by

$$\text{tape-config}_{j,i+1}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle) := \begin{cases} (\otimes^{\bar{p}_j''-1} \tilde{w}_j \otimes^{\ell-\bar{p}_j''+1}, \bar{p}_j'', \bar{p}_j'') & \text{if } \bar{p}_j'' \leq \bar{p}_j'', \\ (\varepsilon, \bar{p}_j'', \bar{p}_j'') & \text{otherwise.} \end{cases}$$

Notice that the situation  $\hat{p}_j^{\parallel} > \tilde{p}_j^{\parallel}$  can occur if  $j \neq j_0$ , and the head of tape  $j$  does not move during the computation  $\gamma_1, \dots, \gamma_v$  of  $T$ ; or if the head reversal takes place while the head is on the block boundary, i.e., if  $p_{j_0}'' = \hat{p}_{j_0}^{\parallel} - 1$ , in which case we have  $\tilde{p}_{j_0}^{\parallel} = \hat{p}_{j_0}^{\parallel} + 1$  by the above definitions.

**ad Case 3:** In this case,  $\gamma_v$  is a final configuration (i.e.,  $q''$  is a final state of the Turing machine  $T$ ). Moreover, on all tapes  $j \in \{1, \dots, t\}$ , we know that throughout  $T$ 's computation  $\gamma_1, \dots, \gamma_v$ , head  $j$  neither changes its direction nor crosses one of the borders of the current block on tape  $j$ . We treat all these tapes in precisely the same way as in case 1 above. In particular, we let

$$\alpha_{|K_{i+1}}(\kappa, c) := (b, e_1'', \dots, e_t'') \quad \text{with} \quad b := (\hat{q}'', \bar{p}_1'', \dots, \bar{p}_t''),$$

where

$$\hat{q}'' := (q'', p_{t+1}'', \dots, p_{t+u}'', w_{t+1}'', \dots, w_{t+u}'')$$

and for all  $j \in \{1, \dots, t\}$ ,

$$\begin{aligned} \bar{p}_j'' &:= \begin{cases} (p_j'', p_j'', \hat{p}_j^{\parallel}, \text{head-direction}_j), & \text{if } \text{head-direction}_j = +1 \\ (\hat{p}_j^{\parallel}, p_j'', p_j'', \text{head-direction}_j), & \text{if } \text{head-direction}_j = -1, \end{cases} \\ e_j'' &:= (\text{head-direction}_j, \text{false}). \end{aligned}$$

For each  $j \in \{1, \dots, t\}$ , we also let

$$\text{tape-config}_{j,i+1}(\langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle) := \begin{cases} (\otimes^{\tilde{p}_j^{\parallel}-1} \tilde{w}_j \otimes^{\ell-\tilde{p}_j^{\parallel}+1}, \tilde{p}_j^{\parallel}, \tilde{p}_j^{\parallel}) & \text{if } \tilde{p}_j^{\parallel} \leq \hat{p}_j^{\parallel}, \\ (\varepsilon, \tilde{p}_j^{\parallel}, \tilde{p}_j^{\parallel}) & \text{otherwise,} \end{cases}$$

where

$$(\tilde{p}_j^{\parallel}, \tilde{p}_j^{\parallel}) := \begin{cases} (\hat{p}_j^{\parallel}, p_j'' - 1) & \text{if } \text{head-direction}_j = +1 \\ (p_j'' + 1, \hat{p}_j^{\parallel}) & \text{if } \text{head-direction}_j = -1, \end{cases}$$

and

$$\tilde{w}_j := w_{j, \tilde{p}_j^{\parallel}}'' \cdots w_{j, \tilde{p}_j^{\parallel}}''.$$

Notice that since  $\gamma_v$  is *final*,  $b$  is a *final* state of the NLM  $M$ , and  $M$ 's run accepts if, and only if, the simulated Turing machine run accepts (recall the definition of  $M$ 's set of final and accepting states at the end of Step 3).

Altogether, this completes the induction step, and we are ready to fix  $M$ 's state set  $A$  and transition function  $\alpha$  as follows:

$$\begin{aligned} A &:= \bigcup_{i \geq 0} A_i, \\ K &:= \bigcup_{i \geq 0} K_i, \\ \alpha &:= \bigcup_{i \geq 0} \alpha_{|K_i}. \end{aligned}$$

Note that

1.  $\alpha$  is well-defined, because  $\alpha_{|K_i}$  and  $\alpha_{|K_{i'}}$  operate identically on all elements in  $(K_i \cap K_{i'}) \times C$  (for all  $i, i' \geq 0$ ).
2.  $K$  consists of all situations  $(a, y_1, \dots, y_t) \in (A \setminus B) \times (\mathbf{A})^t$  that may occur in runs of  $M$ .
3.  $\alpha$  remains undefined for elements  $(a, y_1, \dots, y_t)$  in  $(A \setminus B) \times (\mathbf{A})^t$  that do *not* belong to  $K$ . This is fine, because such a situation  $(a, y_1, \dots, y_t)$  can never occur in an actual run of  $M$ .

This completes Step 4. ◻

Note also that the NLM  $M$  is now fully specified. Due to the construction we know that  $M$  is  $(r, t)$ -bounded, because it has  $t$  lists and the number of head reversals during each run on an input  $\bar{v} = (v_1, \dots, v_m) \in I^m$  is bounded by the number  $r-1 = r(m \cdot (n+1)) - 1$  of head reversals of the according run of the Turing machine  $T$  on input  $v_1\#\dots\#v_m\#$ .

**Step 5:** For every input  $\bar{v} = (v_1, \dots, v_m) \in I^m$  we have

$$\Pr(M \text{ accepts } \bar{v}) = \Pr(T \text{ accepts } v_1\#\dots\#v_m).$$

*Proof:* Let  $\ell_M \in \mathbb{N}$  be an upper bound on the length of runs of the NLM  $M$  (such a number  $\ell_M$  exists, because  $M$  is  $(r, t)$ -bounded; see Lemma 4.8 (a)). For the remainder of this proof we fix an input  $\bar{v} = (v_1, \dots, v_m) \in I^m$  for the NLM  $M$  and we let  $\tilde{v} := v_1\#\dots\#v_m\#$  denote the corresponding input for the Turing machine  $T$ .

From Lemma 5.3 we know that

$$\Pr(T \text{ accepts } \tilde{v}) = \frac{|\{\bar{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \bar{c}_T) \text{ accepts}\}|}{|C_T^\ell|} = \frac{|\{\bar{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \bar{c}_T) \text{ accepts}\}|}{|C|}.$$

Furthermore, we know from Lemma 4.4 that

$$\Pr(M \text{ accepts } \bar{v}) = \frac{|\{\bar{c} \in C^{\ell_M} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}|}{|C|^{\ell_M}}.$$

For showing that  $\Pr(M \text{ accepts } \bar{v}) = \Pr(T \text{ accepts } \tilde{v})$  it therefore suffices to show that

$$|\{\bar{c} \in C^{\ell_M} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}| = |C|^{\ell_M-1} \cdot |\{\bar{c}_T \in C_T^\ell : \rho_T(\tilde{v}, \bar{c}_T) \text{ accepts}\}|.$$

Consequently, it suffices to show that there is a function

$$f : C^{\ell_M} \rightarrow C_T^\ell$$

such that

- for every  $\bar{c} \in C^{\ell_M}$ , the list machine run  $\rho_M(\bar{v}, \bar{c})$  simulates the Turing machine run  $\rho_T(\tilde{v}, f(\bar{c}))$ , and
- for every  $\bar{c}_T \in C_T^\ell$ ,

$$|\{\bar{c} \in C^{\ell_M} : f(\bar{c}) = \bar{c}_T\}| = |C|^{\ell_M-1}. \quad (5.5)$$

We can define such a function  $f$  as follows:

For every sequence

$$\bar{c} = (c^{(1)}, \dots, c^{(\ell_M)}) \in C^{\ell_M},$$

following the construction of the NLM  $M$  in Steps 1–4, we obtain for each  $i \in \{1, \dots, \ell_M\}$  that there is a uniquely defined prefix  $\tilde{c}^{(i)}$  of  $M$ 's nondeterministic choice

$$c^{(i)} = (c_1^{(i)}, \dots, c_\ell^{(i)}) \in C = C_T^\ell,$$

such that the following is true for

$$\tilde{c} := \tilde{c}^{(1)}\tilde{c}^{(2)}\dots\tilde{c}^{(\ell_M)},$$

viewed as a sequence of elements from  $C_T$ :

- (1) The list machine run  $\rho_M(\bar{v}, \bar{c})$  simulates the Turing machine run  $\rho_T(\tilde{v}, \tilde{c})$ , where  $M$  uses in its  $i$ -th step exactly the  $\tilde{c}^{(i)}$ -portion of  $c^{(i)}$  for simulating the according Turing machine steps.
- (2) If  $\tilde{\ell} \leq \ell$  denotes the length of the run  $\rho_T(\tilde{v}, \tilde{c}) = (\rho_1, \dots, \rho_{\tilde{\ell}})$ , then  $\tilde{c}$  has exactly the length  $\tilde{\ell}-1$ .

Now let  $i_0$  denote the maximum element from  $\{1, \dots, \ell_M\}$  such that  $|\tilde{c}^{(i_0)}| \neq 0$  (in particular, this implies that  $\tilde{c} = \tilde{c}^{(1)} \dots \tilde{c}^{(i_0)}$ ). We let  $\tilde{c}^{(i_0)}$  be the prefix of  $c^{(i_0)}$  of length  $\ell - (\tilde{\ell} - 1 - |\tilde{c}^{(i_0)}|)$  and define

$$\tilde{\tilde{c}} := \tilde{c}^{(1)} \dots \tilde{c}^{(i_0-1)} \tilde{c}^{(i_0)}.$$

Note that, viewed as a sequence of elements from  $C_T$ ,  $\tilde{\tilde{c}}$  has length exactly  $\ell$ , and therefore, we can well define

$$f(\tilde{c}) := \tilde{\tilde{c}}.$$

Furthermore, to see that (5.5) is satisfied, note that  $f$  is surjective, i.e., for every  $\tilde{\tilde{c}} \in C_T^\ell$  there exists a  $\tilde{c}$  with  $f(\tilde{c}) = \tilde{\tilde{c}}$ , and

$$|\{\tilde{c} \in C^{\ell_M} : f(\tilde{c}) = \tilde{\tilde{c}}\}| = |C_T|^{\ell \cdot \ell_M - \ell} = |C_T|^{\ell \cdot (\ell_M - 1)} = |C|^{\ell_M - 1}.$$

(For the first equation, note that through  $\tilde{\tilde{c}}$ , exactly  $\ell$  of the possible  $\ell \cdot \ell_M$   $C_T$ -components of  $\tilde{c}$  are fixed, whereas each of the remaining  $\ell \cdot \ell_M - \ell$  components may carry an arbitrary element from  $C_T$ .)

This completes Step 5. ←

Altogether, the proof of Lemma 5.1 is complete. □

## 6 Lower Bounds for List Machines

This section's main result is that it provides constraints on a list machine's parameters, which ensure that list machines which comply to these constraints can neither solve the *multiset equality problem* nor the *checksort problem*. In fact, we prove a slightly stronger result stating that not even the restrictions of the problems to inputs ordered in a particular way can be solved by list machines which comply to the constraints.

**Definition 6.1 (subsequence).** A sequence  $(s_1, \dots, s_\lambda)$  is a *subsequence* of a sequence  $(s'_1, \dots, s'_{\lambda'})$ , if there exist indices  $j_1 < \dots < j_\lambda$  such that  $s_1 = s'_{j_1}, s_2 = s'_{j_2}, \dots, s_\lambda = s'_{j_\lambda}$ . ←

**Definition 6.2 (sortedness).** Let  $m \in \mathbb{N}$  and let  $\pi$  be a permutation of  $\{1, \dots, m\}$ . We define *sortedness*( $\pi$ ) to be the length of the longest subsequence of  $(\pi(1), \dots, \pi(m))$  that is sorted in either ascending or descending order (i.e., that is a subsequence of  $(1, \dots, m)$  or of  $(m, \dots, 1)$ ). ←

The well-known Erdős-Szekeres Theorem [10] implies that the sortedness of every permutation of  $\{1, \dots, m\}$  is at least  $\lfloor \sqrt{m} \rfloor$ . It is easy to construct a permutation showing that this lower bound is tight; for the reader's convenience we give an example: Assume first that  $m = \ell^2$  for some positive integer  $\ell$ . Let  $\varphi$  be the permutation with  $\varphi((i-1) \cdot \ell + j) = (\ell - j) \cdot \ell + i$  for  $1 \leq i, j \leq \ell$ . That is,  $\varphi(1), \dots, \varphi(m)$  is the sequence

$$(\ell - 1) \cdot \ell + 1, (\ell - 2) \cdot \ell + 1, \dots, 1, \quad (\ell - 1) \cdot \ell + 2, (\ell - 2) \cdot \ell + 2, \dots, 2, \quad \dots, \quad (\ell - 1) \cdot \ell + \ell, \dots, \ell.$$

It is easy to see that *sortedness*( $\varphi$ ) =  $\ell$ : Think of the numbers as being arranged in an  $\ell \times \ell$ -matrix. Then each increasing subsequence contains at most one entry per row and each decreasing subsequence contains at most one entry per column. If  $m$  is not a square, then we let  $\ell = \lceil \sqrt{m} \rceil$  and obtain a permutation  $\varphi'$  of  $\{1, \dots, \ell^2\}$  with *sortedness*( $\varphi'$ ) =  $\ell$ , of course this also yields a permutation  $\varphi$  of  $\{1, \dots, m\}$  with *sortedness*( $\varphi$ ) =  $\ell$ . For later reference, we state these observations as a lemma:

**Lemma 6.3.** *For every  $m \geq 1$  there exists a permutation  $\varphi$  with *sortedness*( $\varphi$ )  $\leq \lceil \sqrt{m} \rceil$ .* ←

**Lemma 6.4 (Lower Bound for List Machines).**

*Let  $k, m, n, r, t \in \mathbb{N}$  such that  $m$  is a power of 2 and  $t \geq 2$ ,  $m \geq 24 \cdot (t+1)^{4r} + 1$ ,  $k \geq 2m + 3$ ,  $n \geq 1 + (m^2 + 1) \cdot \log(2k)$ . We let  $I := \{0, 1\}^n$ , identify  $I$  with the set  $\{0, 1, \dots, 2^n - 1\}$ , and divide it into  $m$  consecutive intervals  $I_1, \dots, I_m$ , each of length  $2^n/m$ . Let  $\varphi$  be a permutation of  $\{1, \dots, m\}$  with *sortedness*( $\varphi$ )  $\leq \lceil \sqrt{m} \rceil$ , and let  $\mathcal{S} := I_{\varphi(1)} \times \dots \times I_{\varphi(m)} \times I_1 \times \dots \times I_m$ .*

*Then there is no  $(r, t)$ -bounded NLM  $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$  with  $|A| \leq k$  and  $I = \{0, 1\}^n$ , such that for all  $\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{S}$  we have:*

*If  $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$ , then  $\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2}$ ; otherwise  $\Pr(M \text{ accepts } \bar{v}) = 0$ .* ←

It is straightforward to see that the above lemma, in particular, implies that neither the *(multi)set equality problem* nor the *checksort problem* can be solved by list machines with the according parameters.

**Outline of the proof of Lemma 6.4:**

1. Suppose for contradiction that  $M$  is an NLM that meets the lemma's requirements.
2. Observe that there exists an upper bound  $\ell$  on the length of  $M$ 's runs (Lemma 4.8 (a)) and a particular sequence  $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$  of nondeterministic choices (Lemma 6.5), such that for at least half of the inputs  $\bar{v} := (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{I}$  with  $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$ , the particular run  $\rho_M(\bar{v}, \bar{c})$  accepts.  
We let  $\mathcal{I}_{acc, \bar{c}} := \{\bar{v} \in \mathcal{I} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}$  and, from now on, we only consider runs that are generated by the fixed sequence  $\bar{c}$  of nondeterministic choices.
3. Use the notion of the *skeleton* of a run (cf., Definition 6.7), which, roughly speaking, is obtained from a run by replacing every input value  $v_i$  with its index  $i$  and by replacing every nondeterministic choice  $c \in C$  with the wildcard symbol “?”. In particular, the skeleton contains input *positions* rather than concrete input *values*; but given the skeleton together with the concrete input values and the sequence of nondeterministic choices, the entire run of  $M$  can be reconstructed.
4. Now choose  $\zeta$  to be a skeleton that is generated by the run  $\rho_M(\bar{v}, \bar{c})$  for as many input instances  $\bar{v} \in \mathcal{I}_{acc, \bar{c}}$  as possible, and use  $\mathcal{I}_{acc, \bar{c}, \zeta}$  to denote the set of all those input instances.
5. Show that, throughout its computation,  $M$  can “mix” the relative order of its input values only to a rather limited extent (cf., Lemma 6.13). This can be used to show that for every run of  $M$  on every input  $(v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{I}$  there must be an index  $i_0$  such that  $v_{i_0}$  and  $v'_{\varphi(i_0)}$  are never compared throughout this run.
6. Thus for the specific skeleton  $\zeta$ , there must be an index  $i_0$  such that for all inputs from  $\mathcal{I}_{acc, \bar{c}, \zeta}$ , the values  $v_{i_0}$  and  $v'_{\varphi(i_0)}$  (i.e., the values from the input positions  $i_0$  and  $m + \varphi(i_0)$ ) are never compared throughout the run that has skeleton  $\zeta$ . To simplify notation let us henceforth assume without loss of generality that  $i_0 = 1$ .
7. Now fix  $(v_2, \dots, v_m)$  such that the number of  $v_1$  with

$$V(v_1) := (v_1, v_2, \dots, v_m, v_{\varphi^{-1}(1)}, \dots, v_{\varphi^{-1}(m)}) \in \mathcal{I}_{acc, \bar{c}, \zeta}$$

is as large as possible.

8. Argue that, for our fixed  $(v_2, \dots, v_m)$ , there must be at least two distinct  $v_1$  and  $w_1$  such that  $V(v_1) \in \mathcal{I}_{acc, \bar{c}, \zeta}$  and  $V(w_1) \in \mathcal{I}_{acc, \bar{c}, \zeta}$ . This is achieved by observing that the number of skeletons depends on the machine's parameters  $t, r, m, k$ , but *not* on  $n$  (Lemma 6.9) and by using the lemma's assumption on the machine's parameters  $t, r, m, k, n$ .
9. Now we know that the input values of  $V(v_1)$  and  $V(w_1)$  coincide on all input positions except 1 and  $m + \varphi(1)$ . From 5 we know that the values from the positions 1 and  $m + \varphi(1)$  are never compared throughout  $M$ 's (accepting) runs  $\rho_M(V(v_1), \bar{c})$  and  $\rho_M(V(w_1), \bar{c})$ . (Recall that we assume  $i_0 = 1$  in 6.) From this we obtain (cf., Lemma 6.11) an accepting run  $\rho_M(\bar{u}, \bar{c})$  of  $M$  on input

$$\begin{aligned} \bar{u} &:= (u_1, \dots, u_m, u'_1, \dots, u'_m) \\ &:= (v_1, v_2, \dots, v_m, w_{\varphi^{-1}(1)}, w_{\varphi^{-1}(2)}, \dots, w_{\varphi^{-1}(m)}). \end{aligned}$$

In particular, this implies that  $\Pr(M \text{ accepts } \bar{u}) > 0$ . However, for this particular input  $\bar{u}$  we know that  $u_1 = v_1 \neq w_1 = u_{\varphi(1)}$ , and therefore,  $(u_1, \dots, u_m) \neq (u'_{\varphi(1)}, \dots, u'_{\varphi(m)})$ . This gives us a contradiction to the assumption that  $\Pr(M \text{ accepts } \bar{v}) = 0$  for all inputs  $\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m)$  with  $(v_1, \dots, v_m) \neq (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$ .

The proof of Lemma 6.4 now proceeds as follows. After pointing out an easy observation concerning randomized list machines in Subsection 6.1, we formally fix the notion of the *skeleton* of a list machine's run in Subsection 6.2. Then, in Subsection 6.3 we use the notion of skeleton to show the possibility of composing different runs. Afterwards, in Subsection 6.4, we take a closer look at the information flow that can occur during a list machine's computation, and we show that only a small number of input positions can be compared during an NLM's run. Finally, in Subsection 6.5, we prove Lemma 6.4.

## 6.1 An Easy Observation Concerning Randomized List Machines

**Lemma 6.5.** *Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM, let  $\ell$  be an upper bound on the length of  $M$ 's runs, and let  $\mathcal{J} \subseteq I^m$  such that  $\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2}$ , for all inputs  $\bar{v} \in \mathcal{J}$ . Then there is a sequence  $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$  such that the set*

$$\mathcal{J}_{acc, \bar{c}} := \{\bar{v} \in \mathcal{J} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}$$

has size  $|\mathcal{J}_{acc, \bar{c}}| \geq \frac{1}{2} \cdot |\mathcal{J}|$ .

*Proof:* The proof is a straightforward double counting argument: By assumption we know that

$$\sum_{\bar{v} \in \mathcal{J}} \Pr(M \text{ accepts } \bar{v}) \geq |\mathcal{J}| \cdot \frac{1}{2}.$$

From Lemma 4.4 we obtain

$$\sum_{\bar{v} \in \mathcal{J}} \Pr(M \text{ accepts } \bar{v}) = \sum_{\bar{v} \in \mathcal{J}} \frac{|\{\bar{c} \in C^\ell : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}|}{|C^\ell|}.$$

Therefore,

$$\sum_{\bar{v} \in \mathcal{J}} |\{\bar{c} \in C^\ell : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}| \geq |C^\ell| \cdot \frac{|\mathcal{J}|}{2}.$$

On the other hand,

$$\sum_{\bar{v} \in \mathcal{J}} |\{\bar{c} \in C^\ell : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}| = \sum_{\bar{c} \in C^\ell} |\{\bar{v} \in \mathcal{J} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}|.$$

Consequently,

$$\sum_{\bar{c} \in C^\ell} |\{\bar{v} \in \mathcal{J} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}| \geq |C^\ell| \cdot \frac{|\mathcal{J}|}{2}.$$

Therefore, there must exist at least one  $\bar{c} \in C^\ell$  with

$$|\{\bar{v} \in \mathcal{J} : \rho_M(\bar{v}, \bar{c}) \text{ accepts}\}| \geq \frac{|\mathcal{J}|}{2},$$

and the proof of Lemma 6.5 is complete.  $\square$

## 6.2 Skeletons of list machine runs

To prove lower bound results for list machines, we use the notion of a *skeleton* of a run. Basically, a skeleton describes the information *flow* during a run, in the sense that it does *not* describe the exchanged data items (i.e., input values), but instead, it describes which input *positions* the data items originally came from. The input positions of an NLM  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  are simply the indices  $i \in \{1, \dots, m\}$ .

The following additional notions are needed to define skeletons.

**Definition 6.6 (local\_views( $\rho$ ), ndet\_choices( $\rho$ )).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM.

(a) The *local view*,  $lv(\gamma)$ , of a configuration  $\gamma = (a, p, d, X)$  of  $M$  is defined via

$$lv(\gamma) := (a, d, y) \quad \text{with} \quad y := \begin{pmatrix} x_{1, p_1} \\ \vdots \\ x_{t, p_t} \end{pmatrix}.$$

I.e.,  $lv(\gamma)$  carries the information on  $M$ 's current state, head directions, and contents of the list cells currently being seen.

(b) Let  $\rho = (\rho_1, \dots, \rho_\ell)$  be a run of  $M$ . We define

(i)  $\text{local\_views}(\rho) := (lv(\rho_1), \dots, lv(\rho_\ell))$ .

(ii)  $\text{ndet\_choices}(\rho) \subseteq C^{\ell-1}$  to be the set of all sequences  $\bar{c} = (c_1, \dots, c_{\ell-1})$  such that, for all  $i < \ell$ ,  $\rho_{i+1}$  is the  $c_i$ -successor of  $\rho_i$ .

Note that  $\Pr(\rho) = \frac{|\text{ndet\_choices}(\rho)|}{|C|^{\ell-1}}$ . ⊣

**Definition 6.7 (Index Strings and Skeletons).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM, let  $\bar{v} = (v_1, \dots, v_m) \in I^m$  be an input for  $M$ , let  $\rho$  be a run of  $M$  for input  $\bar{v}$ , and let  $\gamma = (a, p, d, X)$  be one of the configurations in  $\rho$ .

(a) For every cell content  $x_{\tau, j}$  in  $X$  (for each list  $\tau \in \{1, \dots, t\}$ ), we write

$$\text{ind}(x_{\tau, j})$$

to denote the *index string*, i.e., the string obtained from  $x_{\tau, j}$  by replacing each occurrence of input number  $v_i$  by its index (i.e., input position)  $i \in \{1, \dots, m\}$ , and by replacing each occurrence of a nondeterministic choice  $c \in C$  by the wildcard symbol “?”.

(b) For  $y = (x_{1, p_1}, \dots, x_{t, p_t})^\top$  we let

$$\text{ind}(y) := (\text{ind}(x_{1, p_1}), \dots, \text{ind}(x_{t, p_t}))^\top.$$

(c) The *skeleton* of a configuration  $\gamma$ 's local view  $lv(\gamma) = (a, d, y)$  is defined via

$$\text{skel}(lv(\gamma)) := (a, d, \text{ind}(y)).$$

(d) The *skeleton of a run*  $\rho = (\rho_1, \dots, \rho_\ell)$  of  $M$  is defined via

$$\text{skel}(\rho) := (s, \text{moves}(\rho)),$$

where  $s = (s_1, \dots, s_\ell)$  with  $s_1 := \text{skel}(lv(\rho_1))$ , and for all  $i < \ell$ , if  $\text{moves}(\rho) = (\text{move}_1, \dots, \text{move}_{\ell-1})^\top$ ,

$$s_{i+1} := \begin{cases} \text{skel}(lv(\rho_{i+1})) & \text{if } \text{move}_i \neq (0, 0, \dots, 0)^\top \\ \text{“?”} & \text{otherwise.} \end{cases}$$

**Remark 6.8.** Note that, given an input instance  $\bar{v}$  for an NLM  $M$ , the skeleton  $\zeta := \text{skel}(\rho)$  of a run  $\rho$  of  $M$  on input  $\bar{v}$ , and a sequence  $\bar{c} \in \text{ndet\_choices}(\rho)$ , the entire run  $\rho$  can be reconstructed. ⊣

**Lemma 6.9 (Number of Skeletons).**

Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an  $(r, t)$ -bounded NLM with  $t \geq 2$  and  $k := |A| \geq 2$ .

The number

$$|\{\text{skel}(\rho) : \rho \text{ is a run of } M\}|$$

of skeletons of runs of  $M$  is

$$\leq (m + k + 3)^{12 \cdot m \cdot (t+1)^{2r+2} + 24 \cdot (t+1)^r}.$$

*Proof:* We first count the number of skeletons of local views of configurations  $\gamma$  of  $M$ . Let  $\gamma$  be a configuration of  $M$ , and let  $lv(\gamma)$  be of the form  $(a, d, y)$ . Then,

$$skel(lv(\gamma)) = (a, d, ind(y)),$$

where  $a \in A$ ,  $d \in \{-1, 1\}^t$ , and  $ind(y)$  is a string over the alphabet

$$\{1, \dots, m\} \cup \{\text{"?"}\} \cup A \cup \{\langle, \rangle\}.$$

Due to Lemma 4.6 (b), the string  $ind(y)$  has length  $\leq 11 \cdot t^r$ . Therefore,

$$|\{skel(lv(\gamma)) : \gamma \text{ is a configuration of } M\}| \leq k \cdot 2^t \cdot (m+k+3)^{11 \cdot t^r}. \quad (6.1)$$

From Lemma 4.8 we know that for every run  $\rho = (\rho_1, \dots, \rho_\ell)$  of  $M$  there is a number  $\mu \leq (t+1)^{r+1} \cdot m$  and indices  $1 \leq j_1 < j_2 < \dots < j_\mu < \ell$  such that for moves  $(\rho) = (\text{move}_1, \dots, \text{move}_{\ell-1})$  we have:

(i) For every  $i \in \{1, \dots, \ell-1\}$ ,  $\text{move}_i \neq (0, 0, \dots, 0)^\top \iff i \in \{j_1, \dots, j_\mu\}$ .

(ii) If  $\mu = 0$ , then  $\ell \leq k$ .

Otherwise,  $j_1 \leq k$ ;  $j_{v+1} - j_v \leq k$ , for every  $v \in \{1, \dots, \mu-1\}$ ; and  $\ell - j_\mu \leq k$ .

The total number of possibilities of choosing such  $\mu, \ell, j_1, \dots, j_\mu$  is

$$\leq \sum_{\mu=0}^{(t+1)^{r+1} \cdot m} k^{\mu+1} \leq k^{2+(t+1)^{r+1} \cdot m}. \quad (6.2)$$

For each fixed  $\rho$  with parameters  $\mu, \ell, j_1, \dots, j_\mu$ ,  $skel(\rho) = (s, \text{moves}(\rho))$  is of the following form: For every  $i \leq \ell$  with  $i \notin \{j_1, \dots, j_\mu\}$ ,  $\text{move}_i = (0, 0, \dots, 0)^\top$  and  $s_{i+1} = \text{"?"}$ . For the remaining indices  $j_1, \dots, j_\mu$ , there are

$$\leq 3^{t \cdot \mu} \leq 3^{(t+1)^{r+2} \cdot m} \quad (6.3)$$

possibilities of choosing  $(\text{move}_{j_1}, \dots, \text{move}_{j_\mu}) \in (\{0, 1, -1\}^t)^\mu$ , and there are

$$\leq |\{skel(lv(\gamma)) : \gamma \text{ is a configuration of } M\}|^\mu \leq (k \cdot 2^t \cdot (m+k+3)^{11 \cdot t^r})^{(t+1)^{r+1} \cdot m} \quad (6.4)$$

possibilities of choosing  $(s_{j_1+1}, \dots, s_{j_\mu+1}) = (skel(lv(\rho_{j_1+1})), \dots, skel(lv(\rho_{j_\mu+1})))$ .

In total, by computing the product of the terms in (6.2), (6.3), and (6.4), we obtain that the number  $|\{skel(\rho) : \rho \text{ is a run of } M\}|$  of skeletons of runs of  $M$  is at most

$$\begin{aligned} & \left( k^{2+(t+1)^{r+1} \cdot m} \right) \cdot \left( 3^{(t+1)^{r+2} \cdot m} \right) \cdot \left( (k \cdot 2^t \cdot (m+k+3)^{11 \cdot t^r})^{(t+1)^{r+1} \cdot m} \right) \\ & \leq \left( k \cdot 3 \cdot k \cdot 2^t \cdot (m+k+3)^{11 \cdot t^r} \right)^{2+(t+1)^{r+2} \cdot m} \\ & \leq \left( k^2 \cdot 2^{t+\log 3} \cdot (m+k+3)^{11 \cdot t^r} \right)^{2+(t+1)^{r+2} \cdot m}. \end{aligned} \quad (6.5)$$

Obviously,

$$k^2 \leq (k+m+3)^2.$$

Since  $(k+m+3) \geq 2^2$ , we have

$$2^{t+\log 3} \leq (m+k+3)^{t+1}.$$

Inserting this into (6.5), we obtain that the number of skeletons of runs of  $M$  is

$$\begin{aligned} & \leq (m+k+3)^{(11t^r+t+3) \cdot (2+(t+1)^{r+2} \cdot m)} \\ & \leq (m+k+3)^{(12 \cdot (t+1)^r) \cdot (2+(t+1)^{r+2} \cdot m)} \\ & \leq (m+k+3)^{24 \cdot (t+1)^r + 12 \cdot (t+1)^{2r+2} \cdot m}. \end{aligned}$$

This completes the proof of Lemma 6.9.  $\square$

### 6.3 Composition of list machine runs

Different runs of a list machine can be composed as follows:

**Definition 6.10.** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM and let

$$\zeta = ((s_1, \dots, s_\ell), (\text{move}_1, \dots, \text{move}_{\ell-1}))$$

be the skeleton of a run  $\rho$  of  $M$ . We say that two input positions  $i, i' \in \{1, \dots, m\}$ , are *compared* in  $\zeta$  (respectively, in  $\rho$ ) iff there is a  $j \leq \ell$  such that  $s_j$  is of the form

$$\text{skel}(lv(\gamma)) = (a, d, \text{ind}(y)), \quad \text{for some configuration } \gamma,$$

and both  $i$  and  $i'$  occur in  $\text{ind}(y)$ . ⊣

**Lemma 6.11 (Composition Lemma).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM and let  $\ell \in \mathbb{N}$  be an upper bound on the length of  $M$ 's runs. Let  $\zeta$  be the skeleton of a run of  $M$ , and let  $i, i'$  be input positions of  $M$  that are not compared in  $\zeta$ . Let  $\bar{v} = (v_1, \dots, v_m)$  and  $\bar{w} = (w_1, \dots, w_m)$  be two different inputs for  $M$  with

$$w_j = v_j, \quad \text{for all } j \in \{1, \dots, m\} \setminus \{i, i'\}$$

(i.e.,  $\bar{v}$  and  $\bar{w}$  only differ at the input positions  $i$  and  $i'$ ). Furthermore, suppose there exists a sequence  $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$  such that

$$\text{skel}(\rho_M(\bar{v}, \bar{c})) = \text{skel}(\rho_M(\bar{w}, \bar{c})) = \zeta,$$

and  $\rho_M(\bar{v}, \bar{c})$  and  $\rho_M(\bar{w}, \bar{c})$  either both accept or both reject. Then, for the inputs

$$\bar{u} := (v_1, \dots, v_i, \dots, v_{i'-1}, w_{i'}, v_{i'+1}, \dots, v_m)$$

and

$$\bar{u}' := (v_1, \dots, v_{i-1}, w_i, v_{i+1}, \dots, v_{i'}, \dots, v_m)$$

we have

$$\zeta = \text{skel}(\rho_M(\bar{u}, \bar{c})) = \text{skel}(\rho_M(\bar{u}', \bar{c}))$$

and

$$\rho_M(\bar{u}, \bar{c}) \text{ accepts} \iff \rho_M(\bar{u}', \bar{c}) \text{ accepts} \iff \rho_M(\bar{v}, \bar{c}) \text{ accepts} \iff \rho_M(\bar{w}, \bar{c}) \text{ accepts}.$$

*Proof:* Let  $\zeta = ((s_1, \dots, s_\ell), (\text{move}_1, \dots, \text{move}_{\ell-1}))$  be the skeleton as in the hypothesis of the lemma. We show that  $\text{skel}(\rho_M(\bar{u}, \bar{c})) = \zeta$ , and that  $\rho_M(\bar{u}, \bar{c})$  accepts if and only if  $\rho_M(\bar{v}, \bar{c})$  and  $\rho_M(\bar{w}, \bar{c})$  accept. The proof for  $\bar{u}'$  instead of  $\bar{u}$  is the same.

Let  $\text{skel}(\rho_M(\bar{u}, \bar{c})) = ((s'_1, \dots, s'_{\ell''}), (\text{move}'_1, \dots, \text{move}'_{\ell''-1}))$ . Let  $j$  be the maximum index such that

$$(i) \quad (s'_1, \dots, s'_j) = (s_1, \dots, s_j), \text{ and}$$

$$(ii) \quad (\text{move}'_1, \dots, \text{move}'_{j-1}) = (\text{move}_1, \dots, \text{move}_{j-1}).$$

Let  $j'$  be the maximum index such that  $j' \leq j$  and  $s_{j'} = s'_{j'} \neq \text{"?"}$ . By the hypothesis of the lemma we know that  $i$  and  $i'$  do not occur both in  $s_{j'}$ . Thus for some  $\bar{x} \in \{\bar{v}, \bar{w}\}$ ,  $s_{j'}$  contains only input positions where  $\bar{u}$  and  $\bar{x}$  coincide. Let  $\rho_M(\bar{x}, \bar{c}) = (\rho_1, \dots, \rho_{\ell'})$ , and let  $\rho_M(\bar{u}, \bar{c}) = (\rho'_1, \dots, \rho'_{\ell''})$ . Since  $s_{j'}$  contains only input positions where  $\bar{u}$  and  $\bar{x}$  coincide, we have  $lv(\rho_{j'}) = lv(\rho'_{j'})$ . Since  $\text{move}_{j'} = \text{move}'_{j'}$  (because  $(0, \dots, 0)^\top$  for all  $j' \in \{j', \dots, j-1\}$ ), we therefore have  $lv(\rho_j) = lv(\rho'_j)$ . This implies that the behavior in the  $j$ -th step of both runs,  $\rho_M(\bar{x}, \bar{c})$  and  $\rho_M(\bar{u}, \bar{c})$ , is the same.

*Case 1 ( $j = \ell'$ ):* In this case there is no further step in the run, from which we conclude that  $\ell' = \ell''$ . Hence both skeletons,  $\zeta$  and  $\text{skel}(\rho_M(\bar{u}, \bar{c}))$ , are equal. Moreover,  $lv(\rho_j) = lv(\rho'_j)$  implies that both runs either accept or reject.

*Case 2 ( $j < \ell'$ ):* In this case we know that  $\ell'' \geq j+1$ , and that  $\text{move}_j = \text{move}'_j$ . By the choice of  $j$  we also have  $s_{j+1} \neq s'_{j+1}$ , which together with  $\text{move}_j = \text{move}'_j$  implies  $s_{j+1} \neq \text{"?"}$  and  $s'_{j+1} \neq \text{"?"}$ . Let

$s_{j+1} = (a, d, ind)$  and  $s'_{j+1} = (a', d', ind')$ . Since  $lv(\rho_j) = lv(\rho'_j)$ , and the behavior in the  $j$ -th step of both runs is the same, we have  $a = a'$  and  $d = d'$ . So,  $ind$  and  $ind'$  must differ on some component  $\tau \in \{1, \dots, t\}$ . Let  $ind_\tau$  be the  $\tau$ -th component of  $ind$ , and let  $ind'_\tau$  be the  $\tau$ -th component of  $ind'$ .

Since  $(s'_1, \dots, s'_j) = (s_1, \dots, s_j)$  and  $(move'_1, \dots, move'_j) = (move_1, \dots, move_j)$ , the list cells visited directly after step  $j'' \in \{0, \dots, j\}$  of all three runs,  $\rho_M(\bar{v}, \bar{c})$ ,  $\rho_M(\bar{w}, \bar{c})$  and  $\rho_M(\bar{u}, \bar{c})$ , are the same. This in particular implies that  $ind_\tau$  and  $ind'_\tau$  describe the same list cells, though in different runs. So, if  $ind_\tau = \langle p \rangle$  for some input position  $p$ , or  $ind_\tau = \langle \rangle$ , then the cell described by  $ind_\tau$  has not been visited during the first  $j$  steps of all three runs, and therefore,  $ind'_\tau = ind_\tau$ . Now we may assume that  $ind_\tau \neq \langle p \rangle$  for all input positions  $p$ , and  $ind_\tau \neq \langle \rangle$ . Then,  $ind_\tau = \langle a \langle y_1 \rangle \dots \langle y_t \rangle c \rangle$ , where  $(a, d, y_1, \dots, y_t) = s_{j''}$  for some  $j'' \in \{1, \dots, j\}$ , and  $c$  is the  $j''$ -th nondeterministic choice of  $\bar{c}$ . Also,  $ind'_\tau = \langle a' \langle y'_1 \rangle \dots \langle y'_t \rangle c \rangle$ , where  $(a', d', y'_1, \dots, y'_t) = s'_{j''}$ . But  $s_{j''} = s'_{j''}$ , which contradicts  $ind_\tau \neq ind'_\tau$ .

To conclude, only Case 1 can occur, which gives the desired result of the lemma.  $\square$

## 6.4 The information flow during a list machine's run

In this subsection we take a closer look at the information flow that can occur during a list machine's computation and, using this, we show that only a small number of input positions can be compared during an NLM's run. In some sense, the two lemmas of this section form the combinatorial core of the whole proof.

**Definition 6.12.** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM. Let  $\gamma = (a, p, d, X)$  be a configuration of  $M$  with  $X = (x_1, \dots, x_t)^\top$  and  $x_\tau = (x_{\tau,1}, \dots, x_{\tau,m_\tau})$ , for each  $\tau \in \{1, \dots, t\}$ . Furthermore, let  $(i_1, \dots, i_\lambda) \in \{1, \dots, m\}^\lambda$ , for some  $\lambda \in \mathbb{N}$ , be a sequence of input positions.

We say that the sequence  $(i_1, \dots, i_\lambda)$  occurs in configuration  $\gamma$ , if the following is true: There exists a  $\tau \in \{1, \dots, t\}$  and list positions  $1 \leq j_1 \leq \dots \leq j_\lambda \leq m_\tau$  such that, for all  $\mu \in \{1, \dots, \lambda\}$ , the input position  $i_\mu$  occurs in  $ind(x_{\tau, j_\mu})$ .  $\dashv$

The following lemma gives a closer understanding of the information flow that can occur during an NLM's run. Recall that a subsequence  $s'$  of a sequence  $s$  consists of (not necessarily consecutive) entries of  $s$  appearing in  $s'$  in the same order as in  $s$ .

**Lemma 6.13 (Merge Lemma).** Let  $M = (t, m, I, C, A, a_0, \alpha, B, B_{acc})$  be an  $(r, t)$ -bounded NLM, let  $\rho$  be a run of  $M$ , let  $\gamma$  be a configuration in  $\rho$ , and let, for some  $\lambda \in \mathbb{N}$ ,  $(i_1, \dots, i_\lambda) \in \{1, \dots, m\}^\lambda$  be a sequence of input positions that occurs in  $\gamma$ .

Then, there exist  $t^r$  subsequences  $s_1, \dots, s_{t^r}$  of  $(i_1, \dots, i_\lambda)$  such that the following is true, where we let  $s_\mu = (s_{\mu,1}, \dots, s_{\mu,\lambda_\mu})$ , for every  $\mu \in \{1, \dots, t^r\}$ :

$$- \{i_1, \dots, i_\lambda\} = \bigcup_{\mu=1}^{t^r} \{s_{\mu,1}, \dots, s_{\mu,\lambda_\mu}\}, \quad \text{and}$$

- for every  $\mu \in \{1, \dots, t^r\}$ ,  $s_\mu$  is a subsequence either of  $(1, \dots, m)$  or of  $(m, \dots, 1)$ .  $\dashv$

*Proof:* By induction on  $r' \in \{0, \dots, r\}$  we show that for each configuration that occurs during the  $r'$ -th scan (i.e., between the  $(r'-1)$ -st and the  $r'$ -th change of a head direction), the above statement is true for  $t^{r'}$  rather than  $t^r$ .

For the induction start  $r' = 0$  we only have to consider  $M$ 's start configuration. Obviously, every sequence  $(i_1, \dots, i_\lambda)$  that occurs in the start configuration, is a subsequence of  $(1, \dots, m)$ .

For the induction step we note that all that  $M$  can do during the  $r'$ -th scan is *merge* entries from  $t$  different lists produced during the  $(r'-1)$ -st scan. More precisely, let  $\gamma = (a, \bar{p}, \bar{d}, X)$  be the configuration at the beginning of the  $r'$ -th scan, where

$$\bar{p} = (p_1, \dots, p_t), \quad X = (\bar{x}_1, \dots, \bar{x}_t), \quad \text{and} \quad \bar{x}_\tau = (x_{\tau,1}, \dots, x_{\tau,m_\tau}) \quad \text{for each } \tau \in \{1, \dots, t\},$$

and let  $\gamma' = (a', \bar{p}', \bar{d}', X')$  be the configuration at the end of the  $r'$ -th scan, where

$$\bar{p}' = (p'_1, \dots, p'_t), \quad X' = (\bar{x}'_1, \dots, \bar{x}'_t), \quad \text{and} \quad \bar{x}'_\tau = (x'_{\tau,1}, \dots, x'_{\tau,m'_\tau}) \quad \text{for each } \tau \in \{1, \dots, t\}.$$

Then for each  $\tau \in \{1, \dots, t\}$ , we have the following: Let  $j_1, \dots, j_n$  be the sequence of positions on the  $\tau$ -th list visited during the  $r'$ -th scan in that order, i.e., the sequence  $p_\tau, p_\tau + 1, \dots, p'_\tau$  if  $p_\tau \leq p'_\tau$ , and  $p_\tau, p_\tau - 1, \dots, p'_\tau$  otherwise. Then, for each  $\tau' \in \{1, \dots, t\}$ , there is a (non-increasing or non-decreasing) sequence  $j_1^{\tau'}, \dots, j_n^{\tau'}$  of consecutive positions on the  $\tau'$ -th list such that for all  $k \in \{1, \dots, n\}$  there is some  $\hat{a} \in A$  and  $\hat{c} \in C$  with  $x'_{\tau, j_k} = \langle \hat{a} \langle x_{1, j_k^1} \rangle \cdots \langle x_{t, j_k^t} \rangle \hat{c} \rangle$ .

Therefore,  $\{i_1, \dots, i_\lambda\}$  is the union of  $t$  sequences, each of which is a subsequence of either  $(i_1, \dots, i_\lambda)$  or  $(i_\lambda, \dots, i_1)$  (corresponding to a forward scan or a backward scan, respectively), and each of these  $t$  subsequences has been produced during the  $(r'-1)$ -st scan. By induction hypothesis, each of these subsequences is the union of  $t^{r'-1}$  subsequences of  $(1, \dots, m)$  or  $(m, \dots, 1)$ . Consequently,  $(i_1, \dots, i_\lambda)$  must be the union of  $t \cdot t^{r'-1}$  such subsequences.  $\square$

We are now ready to show that only a small number of input positions can be compared during a list machine's run.

**Lemma 6.14 (Only few input positions can be compared by an NLM).**

Let  $M = (t, 2m, I, C, A, a_0, \alpha, B, B_{acc})$  be an NLM with  $2m$  input positions.

Let  $\bar{v} := (v_1, \dots, v_m, v'_1, \dots, v'_m) \in I^{2m}$  be an input for  $M$ , let  $\rho$  be a run of  $M$  on input  $\bar{v}$ , and let  $\zeta := \text{skel}(\rho)$ . Then, for every permutation  $\varphi$  of  $\{1, \dots, m\}$ , there are at most

$$t^{2r} \cdot \text{sortedness}(\varphi)$$

different  $i \in \{1, \dots, m\}$  such that the input positions  $i$  and  $m + \varphi(i)$  are compared in  $\zeta$  (i.e., the input values  $v_i$  and  $v'_{\varphi(i)}$  are compared in  $\rho$ ).

*Proof:* For some  $\lambda \in \mathbb{N}$  let  $i_1, \dots, i_\lambda$  be distinct elements from  $\{1, \dots, m\}$  such that, for all  $\mu \in \{1, \dots, \lambda\}$ , the input positions  $i_\mu$  and  $m + \varphi(i_\mu)$  are compared in  $\zeta$ . From Definition 6.10 and 6.12 it then follows that, for an appropriate permutation  $\pi : \{1, \dots, \lambda\} \rightarrow \{1, \dots, \lambda\}$ , the sequence

$$\iota := (i_{\pi(1)}, m + \varphi(i_{\pi(1)}), i_{\pi(2)}, m + \varphi(i_{\pi(2)}), \dots, i_{\pi(\lambda)}, m + \varphi(i_{\pi(\lambda)}))$$

occurs in some configuration in run  $\rho$ . It is crucial here that a list machine never deletes list entries, but only expands them. Hence if two input positions  $i$  and  $m + \varphi(i)$  are compared in  $\zeta$ , then  $i$  and  $m + \varphi(i)$  appear together in some list cell of every subsequent configuration of the run. From Lemma 6.13 we then obtain that there exist  $t^r$  subsequences  $s_1, \dots, s_{t^r}$  of  $\iota$  such that the following is true, where we let  $s_\mu = (s_{\mu,1}, \dots, s_{\mu,\lambda_\mu})$ , for every  $\mu \in \{1, \dots, t^r\}$ :

$$- \{i_1, \dots, i_\lambda, m + \varphi(i_1), \dots, m + \varphi(i_\lambda)\} = \bigcup_{\mu=1}^{t^r} \{s_{\mu,1}, \dots, s_{\mu,\lambda_\mu}\}, \quad \text{and}$$

- for every  $\mu \in \{1, \dots, t^r\}$ ,  $s_\mu$  is a subsequence either of  $(1, \dots, 2m)$  or of  $(2m, \dots, 1)$ .

In particular, at least one of the sequences  $s_1, \dots, s_{t^r}$  must contain at least  $\lambda' := \lceil \frac{\lambda}{t^r} \rceil$  elements from  $\{i_1, \dots, i_\lambda\}$ . W.l.o.g. we may assume that  $s_1$  is such a sequence, containing the elements  $\{i_1, \dots, i_{\lambda'}\}$ . Considering now the set  $\{m + \varphi(i_1), \dots, m + \varphi(i_{\lambda'})\}$ , we obtain by the same reasoning that one of the sequences  $s_1, \dots, s_{t^r}$  must contain at least  $\lambda'' := \lceil \frac{\lambda'}{t^r} \rceil \geq \frac{\lambda}{t^{2r}}$  elements from  $\{m + \varphi(i_1), \dots, m + \varphi(i_{\lambda'})\}$ . We may assume w.l.o.g. that  $s_2$  is such a sequence, containing the elements  $m + \varphi(i_1), \dots, m + \varphi(i_{\lambda''})$ .

Let us now arrange the elements  $i_1, \dots, i_{\lambda''}, m + \varphi(i_1), \dots, m + \varphi(i_{\lambda''})$  in the same order as they appear in the sequence  $\iota$ . I.e., let  $\pi' : \{1, \dots, \lambda''\} \rightarrow \{1, \dots, \lambda''\}$  be a permutation such that

$$\iota' := (i_{\pi'(1)}, m + \varphi(i_{\pi'(1)}), \dots, i_{\pi'(\lambda'')}, m + \varphi(i_{\pi'(\lambda'')}))$$

is a subsequence of  $\iota$ .

Since  $s_1$  is a subsequence of  $\iota$  and a subsequence of either  $(1, \dots, 2m)$  or  $(2m, \dots, 1)$ , we obtain that

$$\text{either } i_{\pi'(1)} < i_{\pi'(2)} < \dots < i_{\pi'(\lambda'')} \quad \text{or} \quad i_{\pi'(1)} > i_{\pi'(2)} > \dots > i_{\pi'(\lambda'')}.$$

Similarly, since  $s_2$  is a subsequence of  $\iota$  and a subsequence of either  $(1, \dots, 2m)$  or  $(2m, \dots, 1)$ , we obtain that

$$\text{either } m + \varphi(i_{\pi'(1)}) < \dots < m + \varphi(i_{\pi'(\lambda'')}) \quad \text{or} \quad m + \varphi(i_{\pi'(1)}) > \dots > m + \varphi(i_{\pi'(\lambda'')}),$$

and therefore,

$$\text{either } \varphi(i_{\pi'(1)}) < \dots < \varphi(i_{\pi'(\lambda'')}) \quad \text{or} \quad \varphi(i_{\pi'(1)}) > \dots > \varphi(i_{\pi'(\lambda'')}).$$

In other words,  $(\varphi(i_{\pi'(1)}), \dots, \varphi(i_{\pi'(\lambda'')}))$  is a subsequence of  $(\varphi(1), \dots, \varphi(m))$  that is sorted in either ascending or descending order. According to Definition 6.2 we therefore have

$$\lambda'' \leq \text{sortedness}(\varphi).$$

Since  $\lambda'' \geq \frac{\lambda}{t^{2r}}$ , we hence obtain that

$$\lambda \leq t^{2r} \cdot \text{sortedness}(\varphi),$$

and the proof of Lemma 6.14 is complete.  $\square$

## 6.5 Proof of Lemma 6.4

Finally, we are ready for the proof of Lemma 6.4.

Suppose for contradiction that  $M$  is a list machine which meets the requirements of Lemma 6.4. We let

$$\mathcal{I}_{eq} := \{ (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{I} : (v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)}) \}.$$

Note that

$$|\mathcal{I}_{eq}| = \left(\frac{2^n}{m}\right)^m.$$

From the lemma's assumption we know that

$$\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2},$$

for every input  $\bar{v} \in \mathcal{I}_{eq}$ . Our goal is to show that there is some input  $\bar{u} \in \mathcal{I} \setminus \mathcal{I}_{eq}$ , for which there exists an accepting run, i.e., for which  $\Pr(M \text{ accepts } \bar{u}) > 0$ . It should be clear that once having shown this, the proof of Lemma 6.4 is complete.

Since  $M$  is  $(r, t)$ -bounded, we know from Lemma 4.8 that there exists a number  $\ell \in \mathbb{N}$  that is an upper bound on the length of  $M$ 's runs. From Lemma 6.5 we obtain a sequence  $\bar{c} = (c_1, \dots, c_\ell) \in C^\ell$  such that the set

$$\mathcal{I}_{acc, \bar{c}} := \{ \bar{v} \in \mathcal{I}_{eq} : \rho_M(\bar{v}, \bar{c}) \text{ accepts} \}$$

has size

$$|\mathcal{I}_{acc, \bar{c}}| \geq \frac{|\mathcal{I}_{eq}|}{2} \geq \frac{1}{2} \cdot \left(\frac{2^n}{m}\right)^m.$$

Now choose  $\zeta$  to be the skeleton of a run of  $M$  such that the set

$$\mathcal{I}_{acc, \bar{c}, \zeta} := \{ \bar{v} \in \mathcal{I}_{acc, \bar{c}} : \zeta = \text{skel}(\rho_M(\bar{v}, \bar{c})) \}$$

is as large as possible.

**Claim 1**  $|\mathcal{I}_{acc, \bar{c}, \zeta}| \geq \frac{|\mathcal{I}_{acc, \bar{c}}|}{(2k)^{m^2}} \geq \frac{1}{2 \cdot (2k)^{m^2}} \cdot \left(\frac{2^n}{m}\right)^m.$

*Proof:* Let  $\eta$  denote the number of skeletons of runs of  $M$ . From Lemma 6.9 we know that

$$\eta \leq (2m+k+3)^{24 \cdot m \cdot (t+1)^{2r+2} + 24 \cdot (t+1)^r}.$$

From the assumption we know that  $k \geq 2m+3$ , and therefore

$$\eta \leq (2k)^{24 \cdot m \cdot (t+1)^{2r+2} + 24 \cdot (t+1)^r}. \quad (6.6)$$

From the assumption  $m \geq 24 \cdot (t+1)^{4r} + 1$  we obtain that

$$24 \cdot m \cdot (t+1)^{2r+2} + 24 \cdot (t+1)^r \leq 24 \cdot m \cdot (t+1)^{2r+2} + m \leq m^2. \quad (6.7)$$

Altogether, we obtain from (6.6) and (6.7) that

$$\eta \leq (2k)^{m^2}.$$

Since the particular skeleton  $\zeta$  was chosen in such a way that  $|\mathcal{J}_{acc, \bar{c}, \zeta}|$  is as large as possible, and since the total number of skeletons is at most  $(2k)^{m^2}$ , we conclude that

$$|\mathcal{J}_{acc, \bar{c}, \zeta}| \geq \frac{|\mathcal{J}_{acc, \bar{c}}|}{(2k)^{m^2}} \geq \frac{1}{2 \cdot (2k)^{m^2}} \cdot \left(\frac{2^n}{m}\right)^m.$$

Hence, the proof of Claim 1 is complete.  $\square$

**Claim 2** *There is an  $i_0 \in \{1, \dots, m\}$  such that the input positions  $i_0$  and  $m + \varphi(i_0)$  are not compared in  $\zeta$ .*

*Proof:* According to the particular choice of the permutation  $\varphi$  we know that

$$\text{sortedness}(\varphi) \leq \lceil \sqrt{m} \rceil \leq 2\sqrt{m}.$$

Due to Lemma 6.14 it therefore suffices to show that  $m > t^{2r} \cdot 2 \cdot \sqrt{m}$ .

From the assumption that  $m \geq 24 \cdot (t+1)^{4r} + 1$  we know that, in particular,  $m > 4 \cdot t^{4r}$ , i.e.,  $\sqrt{m} > 2 \cdot t^{2r}$ . Hence,  $t^{2r} \cdot 2 \cdot \sqrt{m} < \frac{1}{2} \cdot \sqrt{m} \cdot 2 \cdot \sqrt{m} \leq m$ , and the proof of Claim 2 is complete.  $\square$

Without loss of generality let us henceforth assume that  $i_0 = 1$  (for other  $i_0$ , the proof is analogous but involves uglier notation).

Now choose  $v_2 \in I_{\varphi(2)}, \dots, v_m \in I_{\varphi(m)}$  such that

$$\left| \{ v_1 \in I_{\varphi(1)} : (v_1, v_2, \dots, v_m, v_{\varphi^{-1}(1)}, v_{\varphi^{-1}(2)}, \dots, v_{\varphi^{-1}(m)}) \in \mathcal{J}_{acc, \bar{c}, \zeta} \} \right|$$

is as large as possible. Then, the number of  $v_1$  such that

$$(v_1, v_2, \dots, v_m, v_{\varphi^{-1}(1)}, v_{\varphi^{-1}(2)}, \dots, v_{\varphi^{-1}(m)}) \in \mathcal{J}_{acc, \bar{c}, \zeta}$$

is at least

$$\frac{|\mathcal{J}_{acc, \bar{c}, \zeta}|}{\left(\frac{2^n}{m}\right)^{m-1}} \stackrel{\text{Claim 1}}{\geq} \frac{\left(\frac{2^n}{m}\right)^m}{2 \cdot (2k)^{m^2} \cdot \left(\frac{2^n}{m}\right)^{m-1}} \geq \frac{2^n}{2m \cdot (2k)^{m^2}}.$$

From the assumption we know that  $n \geq 1 + (m^2 + 1) \cdot \log(2k)$ . Therefore,

$$2^n \geq 2 \cdot (2k)^{m^2+1} \geq 2 \cdot (2k) \cdot (2k)^{m^2} \stackrel{k \geq m}{\geq} 2 \cdot 2m \cdot (2k)^{m^2}.$$

Consequently,

$$\frac{2^n}{2m \cdot (2k)^{m^2}} \geq 2.$$

Thus, there are two different elements  $v_1 \neq w_1$  such that for  $(w_2, \dots, w_m) := (v_2, \dots, v_m)$  we have  $\bar{v} := (v_1, \dots, v_m, v_{\varphi^{-1}(1)}, \dots, v_{\varphi^{-1}(m)}) \in \mathcal{I}_{acc, \bar{c}, \zeta}$  and  $\bar{w} := (w_1, \dots, w_m, w_{\varphi^{-1}(1)}, \dots, w_{\varphi^{-1}(m)}) \in \mathcal{I}_{acc, \bar{c}, \zeta}$ . Since the run  $\rho_M(\bar{v}, \bar{c})$  accepts, we obtain from Lemma 6.11 that for the input

$$\bar{u} := (v_1, \dots, v_m, w_{\varphi^{-1}(1)}, \dots, w_{\varphi^{-1}(m)}) \in \mathcal{I} \setminus \mathcal{I}_{eq},$$

the run  $\rho_M(\bar{u}, \bar{c})$  has to accept. Therefore, we have found an input  $\bar{u} \in \mathcal{I} \setminus \mathcal{I}_{eq}$  with

$$\Pr(M \text{ accepts } \bar{u}) > 0.$$

This finally completes the proof of Lemma 6.4.  $\square$

## 7 Lower Bounds for Turing Machines

Finally, we are ready for the proof of the main technical result.

### Proof of Theorem 3.2:

The proof is by a combination of Lemma 5.1 (the Simulation Lemma) and Lemma 6.4 (the lower bound for list machines).

First of all, let us note that without loss of generality we can assume the following:

- (1)  $s(N) \in \Omega(\log N)$   
(this can be enforced by replacing  $s(N)$  by the function  $\max\{s(N), \lceil \log N \rceil\}$ ).
- (2) for  $c(N) := \frac{r(N)}{\log(N/s(N))}$  we have  $\lim_{N \rightarrow \infty} (c(N) \cdot s(N)) = \infty$   
(this can be enforced by replacing  $c(N)$  with the function  $\tilde{c}(N) := \max\{c(N), (1/\sqrt{s(N)})\}$  and by, consequently, replacing  $r(N)$  with the function  $\tilde{c}(N) \cdot \log(N/s(N))$ ).

Furthermore, note that for sufficiently large  $N$ ,

$$24 \cdot (t+1)^{4 \cdot r(N)} + 1 = 2^{\theta(r(N))} = 2^{\theta(c(N) \cdot \log(N/s(N)))} = \left(\frac{N}{s(N)}\right)^{\theta(c(N))} < \left(\frac{N}{s(N)}\right)^{\frac{1}{8}}.$$

The last inequality follows since  $r(N) \in o(\log(N/s(N)))$  and thus  $\lim_{N \rightarrow \infty} c(N) = 0$ .

Hence, there is a  $N_0 \in \mathbb{N}$  such that for all  $N \geq N_0$  we have

$$24 \cdot (t+1)^{4 \cdot r(N)} + 1 \leq \left(\frac{N}{s(N)}\right)^{\frac{1}{8}} \tag{7.1}$$

Next, we need the following:

**Claim 3** For every  $N' \in \mathbb{N}$  there is a natural number  $N \geq N'$  for which there exist  $m, n \in \mathbb{N}$  such that

$$N = 2m \cdot (n+1) \quad \text{and} \quad \left(\frac{N}{s(N)}\right)^{\frac{1}{8}} \leq m \leq \left(\frac{N}{s(N)}\right)^{\frac{2}{8}}$$

Furthermore, the numbers  $N, m, n$  can be chosen in such a way that  $m$  is an even power of 2 (that is,  $m = 2^{2q}$  for some  $q \geq 0$ ).<sup>2</sup>

*Proof:* Recall that  $s(N) = o(N)$ . Choose  $N \geq N'$  such that  $4 \left(\frac{N}{s(N)}\right)^{\frac{1}{8}} \leq \left(\frac{N}{s(N)}\right)^{\frac{2}{8}}$  and such that  $N = 2^p$  for some  $p \geq 1$ . Let  $m$  be an even power of 2 such that  $\left(\frac{N}{s(N)}\right)^{\frac{1}{8}} \leq m \leq \left(\frac{N}{s(N)}\right)^{\frac{2}{8}}$ , say,  $m = 2^{2q}$ . Finally,  $n = 2^{p-1-2q} - 1$ .  $\square$

According to equation (7.1) and Claim 3 we can now choose natural numbers  $m, n, N$  such that

<sup>2</sup>We shall not use  $m$  being an even power of 2 in the present proof, but later in the proof of Theorem 3.4.

- (i)  $N = 2m \cdot (m + 1)$
- (ii)  $\left(\frac{N}{s(N)}\right)^{\frac{1}{8}} \leq m \leq \left(\frac{N}{s(N)}\right)^{\frac{2}{8}}$
- (iii)  $m \geq 24 \cdot (t + 1)^{4 \cdot r(N)} + 1$

Now we are ready to prove the lower bounds for CHECK-SORT and (MULTI)SET-EQUALITY. For contradiction, suppose that there is a  $t \in \mathbb{N}$  such that one of the problems CHECK-SORT, SET-EQUALITY, and MULTISSET-EQUALITY is solved by an  $(r, s, t)$ -bounded  $(\frac{1}{2}, 0)$ -RTM  $T$ . Without loss of generality we may assume that  $t \geq 2$ . By Lemma 5.1 (the Simulation Lemma) there is an  $(r(N), t)$ -bounded nondeterministic list machine  $M$  with

$$k = 2^{O(r(N) \cdot s(N) + \log N)}$$

states that simulates  $T$  on inputs from  $I^{2m}$  for  $I := \{0, 1\}^n$ . By adding dummy states if necessary, we may assume that  $k \geq 2^{r(N) \cdot s(N)}$  and thus, for sufficiently large  $N$ ,

$$k \geq \left(\frac{N}{s(N)}\right)^{c(N) \cdot s(N)} > 2 \cdot \left(\frac{N}{s(N)}\right)^{\frac{2}{8}} + 3 \geq 2m + 3,$$

where the strict inequality holds (for sufficiently large  $N$ ) because  $c(N) \cdot S(N)$  goes to infinity.

Let  $\varphi$  be a permutation of  $\{1, \dots, m\}$  of sortedness at most  $\lceil \sqrt{m} \rceil$ . Then for all instances

$$\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m) \in I_{\varphi(1)} \times \dots \times I_{\varphi(m)} \times I_1 \times \dots \times I_m$$

the following is true:

(\*): If  $(v_1, \dots, v_m) = (v'_{\varphi(1)}, \dots, v'_{\varphi(m)})$ , then  $\Pr(M \text{ accepts } \bar{v}) \geq \frac{1}{2}$ ; otherwise  $\Pr(M \text{ accepts } \bar{v}) = 0$ .

We next argue that the assumptions of Lemma 6.4 (the lower bound for list machines) are met: We already know that  $t \geq 2$ ,  $m \geq 24 \cdot (t + 1)^{4 \cdot r(N)} + 1$ , and  $k \geq 2m + 3$ . For verifying that  $n \geq 1 + (m^2 + 1) \cdot \log(2k)$ , we choose  $\rho$  to be the number with  $\frac{1}{8} \leq \rho \leq \frac{2}{8}$  such that  $m = \left(\frac{N}{s(N)}\right)^\rho$  and observe that

- $(m^2 + 1) \cdot \log(2k) + 1 = \theta(m^2 \cdot \log(2k)) = \theta(r(N) \cdot N^{2\rho} \cdot s(N)^{1-2\rho})$
- $n = \frac{N}{2m} - 1 = \frac{1}{2} \cdot N^{1-\rho} \cdot s(N)^\rho - 1$

Thus,

$$\frac{1 + (m^2 + 1) \cdot \log(2k)}{n} = \theta\left(\frac{r(N) \cdot s(N)^{1-3\rho}}{N^{1-3\rho}}\right) = \theta\left(\frac{r(N)}{\left(\frac{N}{s(N)}\right)^{1-3\rho}}\right) < 1$$

The last inequality is true for sufficiently large  $N$  because  $1 - 3\rho \geq \frac{1}{4}$  and, by assumption,  $r(N) \in o(\log(N/s(N)))$  and thus, in particular,  $r(N) \in o\left(\left(\frac{N}{s(N)}\right)^{1-3\rho}\right)$ . Hence, we have

$$n \geq 1 + (m^2 + 1) \cdot \log(2k).$$

In summary, all the assumptions of Lemma 6.4 are satisfied. Thus, (\*) is a contradiction to Lemma 6.4, and therefore none of the problems CHECK-SORT, SET-EQUALITY, MULTISSET-EQUALITY belongs to  $\text{RST}(r(N), s(N), O(1))$ .

To finish the proof of Theorem 3.2 it remains to prove that the problem SORT does not belong to  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ . Of course, the CHECK-SORT problem can be solved for input  $x_1 \# \dots \# x_m \# y_1 \# \dots \# y_m \#$  by (1) sorting  $x_1 \# \dots \# x_m$  in ascending order and writing the sorted sequence,  $x'_1 \# \dots \# x'_m$  onto the second external memory tape, and (2) comparing  $y_1 \# \dots \# y_m$  and the (sorted) sequence  $x'_1 \# \dots \# x'_m$  in parallel.

Therefore, if the problem SORT was in  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ , i.e., could be solved by an  $(o(\log N), O(N^{1-\varepsilon}), O(1))$ -bounded  $\text{LasVegas-RTM}$   $T$ , then we could solve the CHECK-SORT problem by an  $(o(\log N), O(N^{1-\varepsilon}), O(1))$ -bounded  $(\frac{1}{2}, 0)$ -RTM  $T'$  which uses  $T$  as a subroutine such that  $T'$  rejects whenever  $T$  answers “*I don't know*” and  $T'$  accepts whenever  $T$  produces a sorted sequence that is equal to the sequence  $y_1\#\dots\#y_m$ . This completes the proof of Theorem 3.2.  $\square$

For the proof of Theorem 3.4, we need the following lemma, which follows from the proof above. Recall from the discussion before the Lemma 6.3 that for every square number  $m = \ell^2$  the permutation  $\varphi$  of  $\{1, \dots, m\}$  defined by  $\varphi((i-1)\cdot\ell + j) = (\ell-j)\cdot\ell + i$  for  $1 \leq i, j \leq \ell$  has sortedness  $\sqrt{m}$ . In the following, we denote this permutation by  $\varphi_m$ . For every  $\varepsilon > 0$  we consider the following restriction of the checksort problem:

**CHECK $_{\varphi, \varepsilon}$**

*Instance:*  $v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$ , where  $m \geq 0$  is an even power of 2, and

$$(v_1, \dots, v_m, v'_1, \dots, v'_m) \in I_{\varphi_m(1)} \times \dots \times I_{\varphi_m(m)} \times I_1 \times \dots \times I_m.$$

The sets  $I_1, \dots, I_m$  are obtained as the partition of the set  $I := \{0, 1\}^{m^\delta}$  into  $m$  consecutive subsets, each of size  $2^{m^\delta}/m$ , where  $\delta := \lceil 4/\varepsilon \rceil$

*Problem:* Decide if  $(v_1, \dots, v_m) = (v'_{\varphi_m(1)}, \dots, v'_{\varphi_m(m)})$ .

**Lemma 7.1.** *Let  $\varepsilon$  be constant with  $0 < \varepsilon < 1$ . Let  $r, s: \mathbb{N} \rightarrow \mathbb{N}$  such that  $r(N) \in o(\log N)$  and  $s(N) \in O(N^{1-\varepsilon})$ . Then, there is no  $(r, s, O(1))$ -bounded  $(\frac{1}{2}, 0)$ -RTM that solves CHECK $_{\varphi, \varepsilon}$ .  $\dashv$*

The proof of this lemma is a straightforward adaption of the proof of Theorem 3.2 above.

**Proof of Theorem 3.4:**

Similarly as in the proof of Theorem 3.2, it is easy to see that if SHORT-SORT belongs to  $\text{LasVegas-RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ , then SHORT-CHECK-SORT belongs to  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ .

To prove Theorem 3.4, it thus suffices to prove the desired lower bounds for SHORT-CHECK-SORT, SHORT-SET-EQUALITY, and SHORT-MULTISET-EQUALITY. To obtain these bounds, we reduce the problem CHECK $_{\varphi, \varepsilon}$  to SHORT-CHECK-SORT and SHORT-(MULTI)SET-EQUALITY in such a way that the reduction can be carried out in  $\text{ST}(O(1), O(\log N), 2)$ . More precisely, we construct a reduction (i.e., a function)  $f$  that maps every instance

$$\bar{v} := v_1\#\dots\#v_m\#v'_1\#\dots\#v'_m\#$$

of CHECK $_{\varphi, \varepsilon}$  to an instance

$$f(\bar{v})$$

of SHORT-CHECK-SORT (respectively, of SHORT-SET-EQUALITY or SHORT-MULTISET-EQUALITY), such that

- (1) the string  $f(\bar{v})$  is of length  $\Theta(|\bar{v}|)$ ,
- (2)  $f(\bar{v})$  is a “yes”-instance of SHORT-CHECK-SORT (respectively, a “yes”-instance of SHORT-(MULTI)SET-EQUALITY if, and only if,  $\bar{v}$  is a “yes”-instance of CHECK $_{\varphi, \varepsilon}$ , and
- (3) there is an  $(O(1), O(\log N), 2)$ -bounded deterministic Turing machine that, when given an instance  $\bar{v}$  of CHECK $_{\varphi, \varepsilon}$ , computes  $f(\bar{v})$ .

It should be clear that the existence of such a mapping  $f$  shows that if SHORT-CHECK-SORT (respectively, SHORT-(MULTI)SET-EQUALITY) belongs to the class  $\text{RST}(O(r), O(s), O(1))$ , for some  $s \in \Omega(\log N)$ , then also CHECK $_{\varphi, \varepsilon}$  belongs to  $\text{RST}(O(r), O(s), O(1))$ . If  $r$  and  $s$  are chosen according to the assumption of Theorem 3.4, this would cause a contradiction to Lemma 7.1. Therefore, SHORT-(MULTI)SET-EQUALITY and SHORT-CHECK-SORT do not belong to the class  $\text{RST}(o(\log N), O(N^{1-\varepsilon}), O(1))$ .

Now let us concentrate on the construction of the reduction  $f$ .

By definition of  $\text{CHECK}_{\varphi,\varepsilon}$ , each string  $v_i$  is a 0-1-string of length  $m^\delta$ , where  $\delta = \lceil 4/\varepsilon \rceil$ . For  $i \in \{1, \dots, m\}$ , we subdivide the 0-1-string  $v_i \in \{0, 1\}^{m^\delta}$  into  $\mu := \lceil \frac{m^\delta}{\log m} \rceil$  consecutive blocks  $v_{i,1}, \dots, v_{i,\mu}$ , each of which has length  $\log m$  (to ensure that also the last sub block has length  $\log m$ , we may pad it with leading 0s). In the same way, we subdivide the string  $v'_i$  into sub blocks  $v'_{i,1}, \dots, v'_{i,\mu}$ . For a number  $i \in \{1, \dots, m\}$  we use  $\text{BIN}(i)$  to denote the binary representation of  $i-1$  of length  $\log m$ ; and for a number  $j \in \{1, \dots, \mu\}$  we use  $\text{BIN}'(j)$  to denote the binary representation of  $j-1$  of length  $\delta \cdot \log m$ . For every  $i \in \{1, \dots, m\}$  and  $j \in \{1, \dots, \mu\}$  we let

$$\begin{aligned} w_{i,j} &:= \text{BIN}(\varphi_m(i)) \quad \text{BIN}'(j) \quad v_{i,j}, \\ w'_{i,j} &:= \text{BIN}(i) \quad \text{BIN}'(j) \quad v'_{i,j}, \end{aligned}$$

for every  $i \in \{1, \dots, m\}$  we let

$$\begin{aligned} u_i &:= w_{i,1} \# w_{i,2} \# \dots \# w_{i,\mu} \#, \\ u'_i &:= w'_{i,1} \# w'_{i,2} \# \dots \# w'_{i,\mu} \#, \end{aligned}$$

and finally, we define

$$f(\bar{v}) := u_1 \dots u_m u'_1 \dots u'_m.$$

Clearly,  $f(\bar{v})$  can be viewed as an instance for  $\text{SHORT-CHECK-SORT}$  or  $\text{SHORT-(MULTI)SET-EQUALITY}$ , where  $m' := \mu \cdot m = \lceil \frac{m^{\delta+1}}{\log m} \rceil$  pairs  $w_{i,j}$  and  $w'_{i,j}$  of 0-1-strings of length  $(\delta + 2) \cdot \log m \leq 2 \log m'$  are given. Let us now check that the function  $f$  has the properties (1)–(3).

**ad (1):** Every instance  $\bar{v}$  of  $\text{CHECK}_{\varphi,\varepsilon}$  is a string of length  $N = \Theta(m \cdot m^\delta) = \Theta(m^{\delta+1})$ , and  $f(\bar{v})$  is a string of length  $N' = \Theta(m^{\delta+1})$ .

**ad (2):**

$$\begin{aligned} &\bar{v} \text{ is a "yes"-instance of } \text{CHECK-}\varphi \\ \iff &(v_1, \dots, v_m) = (v'_{\varphi_m(1)}, \dots, v'_{\varphi_m(m)}) \\ \iff &(v_{\varphi_m^{-1}(1)}, \dots, v_{\varphi_m^{-1}(m)}) = (v'_1, \dots, v'_m) \\ \iff &\text{for all } i \in \{1, \dots, m\}, (w_{\varphi_m^{-1}(i),1}, \dots, w_{\varphi_m^{-1}(i),\mu}) = (w'_{i,1}, \dots, w'_{i,\mu}). \end{aligned} \quad (7.2)$$

It is straightforward to see that (7.2) holds if, and only if,  $f(\bar{v})$  is a “yes”-instance of  $\text{SHORT-(MULTI)SET-EQUALITY}$ . Furthermore, as the list of 0-1-strings in the second half of  $f(\bar{v})$  is *sorted* in ascending order,  $f(\bar{v})$  is a “yes”-instance of  $\text{SHORT-CHECK-SORT}$  if, and only if, it is a “yes”-instance of  $\text{SHORT-(MULTI)SET-EQUALITY}$ .

**ad (3):** In a first scan of the input tape, a deterministic Turing machine can compute the number  $m$  and store its binary representation on an internal memory tape.

It is easy to see that the binary representation of  $\varphi_m(i)$  can be computed in space  $O(m)$  and hence entirely in the main memory of our machine. (Here, this is particularly easy because  $m$  is an even power of 2.) Therefore, during a second scan of the input tape, the machine can produce the string  $f(\bar{v})$  on a second external memory tape (without performing any further head reversals on the external memory tapes).

Altogether, the proof of Theorem 3.4 is complete.  $\square$

## 8 A “Rectangle Lemma” for $(r, s, t)$ -bounded Turing Machines

We close the paper by stating a lemma that summarizes the combinatorial essence of the results in sections 4–6. For the precise statement of the lemma, we need the following notation:

Recall from Definition 6.2 for the notion *sortedness*( $\varphi$ ) of a permutation  $\varphi$ . As further notation, we use  $\pi$  to denote the ordinary projection operator from relational algebra. I.e., if  $\bar{v} = (v_1, \dots, v_k)$  is a  $k$ -tuple and  $J = \{j_1, \dots, j_\ell\}$  is a set of indices with  $1 \leq j_1 < \dots < j_\ell \leq k$ , then  $\pi_J(\bar{v}) = (v_{j_1}, \dots, v_{j_\ell})$ . If  $\mathcal{S}$  is a set of  $k$ -tuples, then  $\pi_J(\mathcal{S}) = \{\pi_J(\bar{v}) : \bar{v} \in \mathcal{S}\}$ .

**Definition 8.1 (rectangle).** Let  $1 \leq i < j \leq k$  and let  $\mathcal{S}$  be a set of  $k$ -tuples. Then  $\mathcal{S}$  is an  $(i, j)$ -*rectangle* iff the following two conditions are satisfied:

- (1) All tuples  $\bar{u}$  and  $\bar{v}$  in  $\mathcal{S}$  satisfy  $\pi_{\{1,\dots,k\}\setminus\{i,j\}}(\bar{u}) = \pi_{\{1,\dots,k\}\setminus\{i,j\}}(\bar{v})$ .  
 I.e., all tuples in  $\mathcal{S}$  coincide in all components except for, potentially,  $i$  and  $j$ .
- (2) There exist sets  $X$  and  $Y$  such that  $\pi_{i,j}(\mathcal{S}) = X \times Y$ .  
 Thus, if  $\bar{u} = (u_1, \dots, u_k)$  and  $\bar{v} = (v_1, \dots, v_k)$  are tuples in  $\mathcal{S}$ , then also the tuple  $\bar{u}'$  obtained from  $\bar{u}$  by replacing  $u_j$  with  $v_j$  belongs to  $\mathcal{S}$ .  $\dashv$

**Lemma 8.2 (rectangle lemma).** *Let  $r, s : \mathbb{N} \rightarrow \mathbb{N}$ , let  $t \in \mathbb{N}$ , and let  $T$  be an  $(r, s, t)$ -bounded randomized Turing machine. Let  $\Sigma$  be a finite alphabet such that  $T$  processes input strings from  $\Sigma^*$ . Furthermore, let  $\#$  be an arbitrary symbol in  $\Sigma$ .*

*Then there exists a number  $\lambda \in \mathbb{N}$  such that the following is true for all  $m, n \in \mathbb{N}$ : Letting  $I := (\Sigma \setminus \{\#\})^n$  and  $N := 2m \cdot (n+1)$ , there exist*

- a finite set  $C$  of size  $|C| = 2^{2m \cdot 2^{O(r(N) \cdot s(N) + \lg N)}}$  (the elements in  $C$  are called “nondeterministic choices”),<sup>3</sup>
- a finite set  $S$  of size  $|S| \leq 2^{2m \cdot (r(N) \cdot s(N) + \lg N) \cdot 2^{\lambda \cdot r(N)}}$  (the elements in  $S$  are called “skeletons”),
- a set  $S_{acc} \subseteq S$  (the elements in  $S_{acc}$  are called “accepting skeletons”), and
- a function  $\sigma : I^{2m} \times C \rightarrow S$ ,

such that the following two statements are true:

- (1) For every  $\bar{v} = (v_1, \dots, v_{2m}) \in I^{2m}$  and the string  $\tilde{v} := v_1\#\dots\#v_{2m}\#$ ,

$$\Pr(T \text{ accepts } \tilde{v}) = \frac{|\{c \in C : \sigma(\bar{v}, c) \in S_{acc}\}|}{|C|}$$

- (2) For every permutation  $\varphi$  of  $\{1, \dots, m\}$  and for every  $\zeta \in S$  there exists a set  $J \subseteq \{1, \dots, m\}$  of size

$$|J| \geq m - \text{sortedness}(\varphi) \cdot t^{2 \cdot r(N)}$$

such that for every  $c \in C$ , every  $i \in J$ , and all  $\bar{a} \in I^{2m-2}$  the set

$$\mathcal{S} := \{\bar{v} \in I^{2m} : \sigma(\bar{v}, c) = \zeta \text{ and } \bar{a} = \pi_{\{1,\dots,2m\}\setminus\{i,m+\varphi(i)\}}(\bar{v})\}$$

is an  $(i, m+\varphi(i))$ -rectangle.  $\dashv$

*Proof:* The lemma follows by a straightforward combination of Lemma 5.1, Lemma 4.8 (a), Lemma 6.9, Lemma 6.14, and Lemma 6.11.  $\square$

Note that Lemma 8.2 reduces the problem of proving lower bounds for  $(r, s, t)$ -bounded randomized Turing machines to a purely combinatorial problem. An example of how to use Lemma 8.2 for proving the lower bound stated in Corollary 3.3 can be found in [22]. Lemma 8.2 also serves as a first step in the lower bound proofs of [8].

## References

- [1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS 2004)*, pages 540–549. IEEE Computer Society, 2004.
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
- [3] L. Arge and P. Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. In J. Abello and J. Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 139–159. AMS, 1999.

<sup>3</sup>If  $T$  is deterministic, we even have  $|C| = 1$ .

- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2002)*, pages 1–16. ACM, 2002.
- [5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004)*, pages 177–188. ACM, 2004.
- [6] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2005)*, pages 216–227. ACM, 2005.
- [7] P. Beame and D.-T. Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 499–508. IEEE Computer Society, 2008.
- [8] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 689–698. ACM, 2007.
- [9] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.
- [10] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
- [11] M. Grohe, A. Hernich, and N. Schweikardt. Randomized computations on large data sets: Tight lower bounds. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 243–252. ACM, 2006.
- [12] M. Grohe, C. Koch, and N. Schweikardt. The complexity of querying external memory and streaming data. In *Proceedings of 15th International Symposium on Fundamentals of Computation Theory (FCT 2005)*, volume 3623 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005.
- [13] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theoretical Computer Science*, 380(1-2):199–217, 2007. Special issue for selected papers from ICALP’05.
- [14] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2005)*, pages 238–249. ACM, 2005.
- [15] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series In Discrete Mathematics And Theoretical Computer Science, 1999.
- [16] A. Hernich and N. Schweikardt. Reversal complexity revisited. *Theoretical Computer Science*, 401(1–3):191–205, 2008.
- [17] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
- [18] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [19] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
- [20] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

- [21] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [22] N. Schweikardt. Machine models and lower bounds for query processing. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2007)*, pages 41–52. ACM, 2007.
- [23] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.
- [24] K. Wagner and G. Wechsung. *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, 1986.
- [25] World Wide Web Consortium. XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005, 2005. <http://www.w3.org/TR/xpath20/>.