comput. complex.

DOI 10.1007/s00037-013–0057-1

computational complexity

TRADEOFF LOWER LOUNDS FOR STACK MACHINES

Matei David and
Periklis A. Papakonstantinou

Abstract. A space-bounded Stack Machine is a regular Turing Machine with a read-only input tape, several space-bounded read-write work tapes, and an unbounded stack. Stack Machines with a logarithmic space bound have been connected to other classical models of computation, such as polynomial-time Turing Machines (P) (Cook in J Assoc Comput Mach 18:4–18, 1971) and polynomial size, polylogarithmic depth, bounded fan-in circuits (NC) e.g., Borodin et al. (SIAM J Comput 18, 1989).

In this paper, we present significant new lower bounds and techniques for Stack Machines. This comes in the form of a trade-off lower bound between *space* and *number of passes over the input tape*. Specifically, we give an explicit permuted inner product function such that any Stack Machine computing this function requires either $\Omega\left(N^{1/4-\epsilon}\right)$ space or $\Omega\left(N^{1/4-\epsilon}\right)$ number of passes for every constant $\epsilon > 0$, where N is the input size. In the case of logarithmic space Stack Machines, this yields an *unconditional* $\Omega\left(N^{1/4-\epsilon}\right)$ lower bound for the number of passes. To put this result in perspective, we note that Stack Machines with logarithmic space and a single pass over the input can compute Parity, Majority, as well as certain languages outside NC. The latter follows from Allender (J Assoc Comput Mach 36:912–928, 1989), conditional on the widely believed complexity assumption that $\mathsf{PSPACE} \subsetneq \mathsf{EXP}$.

Our technique is a novel communication complexity reduction, thereby extending the already wide range of models of computation for which communication complexity can be used to obtain lower bounds. Informally, we show that a k-player number-in-hand (NIH) communication protocol for a base function f can efficiently simulate a space- and pass-bounded Stack Machine for a related function F, which consists

Birkhäuser

of several "permuted" instances of f, bundled together by a combining function h. Trade-off lower bounds for Stack Machines then follow from known communication complexity lower bounds.
The framework for this reduction was given by Beame & Huynh-Ngoc (2008), who used it to obtain similar trade-off lower bounds for Turing Machines with a constant number of pass-bounded external tapes. We also prove that the latter cannot efficiently simulate Stack Machines, conditional on the complexity assumption that $\mathsf{E} \not\subseteq \mathsf{PSPACE}$. It is the treatment of an *unbounded* stack which constitutes the main technical novelty in our communication complexity reduction.

Keywords. Turing Machine, stack, AuxPDA, lower bound, communication complexity, space bound, reversals, streaming.

Subject classification. 68Q05.

1. Introduction

One of the goals of complexity theory is understanding the relative power of various models of computation. Consider the classes P and $\mathsf{LOGSPACE}$ of languages decided by deterministic Turing Machines (TMs) in polynomial time and logarithmic space, respectively. Also consider $\mathsf{NC} = \bigcup_{i \geq 0} \mathsf{NC}^i$, where NC^i is the class of languages decided by (uniform) circuits of polynomial size and depth $\mathrm{O}\left((\log n)^i\right)$, consisting of bounded fan-in And, Or, and Not gates. We regard P and $\mathsf{LOGSPACE}$ as modeling efficient time-bounded and space-bounded computation, respectively, and we regard the class NC as modeling efficient parallelizable computation. We know that

$$\mathsf{NC}^1 \subseteq \mathsf{LOGSPACE} \subseteq \mathsf{NC}^2 \subseteq \cdots \subseteq \mathsf{NC}^i \subseteq \cdots \subseteq \mathsf{P},$$

but we do not know whether any of the inclusions are proper.

One possible way to attack the problem of separating complexity classes defined using different models of computation (e.g., Turing Machines, combinatorial circuits) and different resource bounds (e.g., time, space, size, depth) is to rephrase the problem as separations of classes based on a common computational model and a common resource bound and prove lower bounds for that common resource.

Consider the problem of characterizing polynomial-time computation in terms of a space bound. We know that a logarithmic-*space* Turing Machine can be simulated by a polynomial-*time* Turing Machine, and we believe that the opposite is not true in general (i.e., that $\mathsf{LOGSPACE} \subsetneq \mathsf{P}$). Furthermore, we also believe that, for example, some polylogarithmic-space Turing Machine cannot be simulated by a polynomial-time Turing Machine. Thus, there seems to be no obvious way to exactly capture polynomial-time computation in terms of a space bound.

A *stack*, also called a push-down store, models an unlimited storage space that comes with a Last-In First-Out access restriction. A *Stack Machine* (or SM for short) is a classical Turing Machine equipped with a push-down tape. Previous work (see e.g., Cook 1971) makes a distinction between a "stack" and a "push-down memory". In older works a stack is defined as a push-down memory with additional read-only access to its content. We do not make such a distinction here. For us a stack is what was known as a push-down store. In particular, we use the term Stack Machine to refer to what was previously known as deterministic auxiliary push-down automaton (or AuxPDA). A space bound for a Stack Machine refers exclusively to the size of its work tapes and not to its stack. In light of the previous paragraph, Cook (1971) gives a fascinating characterization of a time-bounded complexity class in terms of a space-bounded computational model, showing that the class of languages decided by logarithmic-*space* Stack Machines exactly equals that of languages decided by polynomial-*time* Turing Machines (P).

It is not hard to show that there is no loss in computational power in assuming that a logarithmic space Stack Machine also operates in at most exponential ($2^{n^{\mathrm{O}(1)}}$) *time* (Ruzzo 1980). A series of subsequent results, for example, Allender (1989), Borodin *et al.* (1989), Ruzzo (1980, 1981), Venkateswaran (1991) establish a perhaps surprising connection between simultaneously space- and time-bounded Stack Machines and combinatorial circuits: *non-deterministic* logarithmic-space Stack Machines that run in time $2^{\mathrm{O}\left(\log^i n\right)}$ *precisely* characterize SAC^i, the extension of NC^i in which Not gates are at the input level and we allow Or gates with unbounded fan-in. Furthermore, denoting by $\mathsf{SM}(s,t)$ the class of languages decided by deterministic Stack Machines operating in space s and time t, we now know that, for every integer $i \geq 1$,

$$\begin{aligned}\mathsf{NC}^i \subseteq \mathsf{SM}\left(\mathrm{O}(\log n), 2^{\mathrm{O}\left((\log n)^i\right)}\right) \subseteq \mathsf{NC}^{i+1} \subseteq \cdots \\ \cdots \subseteq \mathsf{P} = \mathsf{SM}\left(\mathrm{O}(\log n), 2^{n^{\mathrm{O}(1)}}\right).\end{aligned}$$

In spite of the connections laid out above between Stack Machines and major open problems in computational complexity, this model is not well understood.

1.1. Our contribution. As the main contribution of this work, we provide involved new lower bounds and techniques specifically for Stack Machines, which significantly improve Brandenburg (1977). We do not know how to tackle time lower bounds directly. Instead, we consider *the number of (two-way) passes a Stack Machine makes over its input*. We assume that a Stack Machine has at least logarithmic space and, without losing generality, that in every pass, the input head moves from one end of the tape to the other. Sometimes this measure is also called reversal complexity, as the number of passes equals one plus the number of reversals. The lower bounds we prove come in the form of trade-off lower bounds between space and number of passes. Specifically, we give two examples of functions for which any Stack Machine requires either $\Omega\left(N^\beta\right)$ space or $\Omega\left(N^\beta\right)$ passes, for some $\beta < 1$. In the case of logarithmic-space Stack Machines, this translates into unconditional $\Omega\left(N^\beta\right)$ lower bounds for the number of passes.

Communication complexity has been used to derive lower bounds in a wide variety of other areas of theoretical computer science: cell probe complexity, VLSI circuit design, Turing Machine complexity, circuit complexity, pseudorandomness, algorithmic game theory, proof complexity, and streaming. Perhaps as important as its corollaries, our main technical contribution is to show that communication complexity can also be used to derive lower bounds for Stack Machines.

Consider a *base* function $f = f_{k,n} : (\{0,1\}^n)^k \to \{0,1\}$ and a *combining* function $h : \{0,1\}^m \to \{0,1\}$ that is symmetric and has a neutral element (e.g., this is the case for OR and XOR, with neutral element 0). Let $F = F_{k,m\cdot n} : ((\{0,1\}^n)^m)^k \to \{0,1\}$ be a *lifted* function which consists, informally, of m instances of f on disjoint inputs, "permuted" in a way made precise in Section 3, and "glued together" by h. Our main technical contribution is the following reduction, saying that a k-player number-in-hand communication protocol for f can efficiently simulate a Stack Machine for F.

THEOREM 1.1. *Let $k = k(n) \geq 2$ be a non-decreasing function and let $m = m(n) \geq 1$ be a non-decreasing function such that $k \leq m^{\mathrm{O}(1)}$. Let $N = N(n) := k \cdot m \cdot n$. Let $s = s(N)$ and $r = r(N)$ be non-decreasing functions and let $\delta < 1/2$ be a constant. Let $d := k \cdot r \cdot \log r/\sqrt{m}$. Let $f = f_{k,n}$ be a boolean base function, and let $F = F_N$ be a function related to f in the sense informally described above and made precise in* Section 3.

Assume there exists a randomized (even, nonuniform) Stack Machine for F with space bound s, pass bound r, and error δ. Then, there exists a k-player number-in-hand randomized protocol for f, with cost $\mathrm{O}\left(k \cdot r \cdot s \cdot \log(k \cdot r)\right)$ and error at most $\delta + \mathrm{O}\left(d\right)$.

As a consequence of Theorem 1.1 and of the known communication complexity lower bound for the Inner Product function (Chor & Goldreich 1988), we obtain the following trade-off lower bound, which is the first of its kind.

COROLLARY 1.2 (Informal statement). *Let $\epsilon > 0$ and $\delta < 1/2$ be constants. There exists an "inner product-like" function $F = F_N \in$ LOGSPACE such that any Stack Machine computing F with error δ requires space $s = \omega\left(N^{1/4-\epsilon}\right)$ or passes $r = \omega\left(N^{1/4-\epsilon}\right)$.*

In particular, a log-space *Stack Machine needs $\omega\left(N^{1/4-\epsilon}\right)$ passes to compute F.*

The ℓ-th frequency moment of a sequence $\bar{a} = (a_1, \ldots, a_t)$, where $a_i \in [R]$, is $\mathrm{Freq}_\ell(\bar{a}) = \sum_{j\in[R]} f_j^\ell$, where $f_j = |\{i \in [t] \mid a_i = j\}|$. Computing Freq_ℓ is a well-studied problem in the streaming literature Alon *et al.* (1999). As another consequence of Theorem 1.1, we also obtain the following result, which can be interpreted as saying that a stack *does not help* in computing frequency moments.

Corollary 1.3. *Let $\ell > 4$, $\epsilon \geq 0$, and $\delta < 1/2$ be constants. There exists a constant $0 < \beta < 1$ such that any randomized Stack Machine computing a $(1+\epsilon)$ multiplicative approximation of* Freq_ℓ *with error δ requires space $s = \omega\left(N'^{\beta}\right)$ or passes $r = \omega\left(N'^{\beta}\right)$, where N' denotes the input size.*

In particular, a log-space *Stack Machine needs $\omega\left(N'^{\beta}\right)$ passes to approximate* Freq_ℓ.

In fact, this corollary holds true even when the Stack Machine has read-write access to its input (but still the number of passes is bounded).

For this result, we use a number-in-hand communication complexity lower bound for the promise Set Intersection function (Gronemeier 2009), along with a streaming reduction between the problems of computing the promise Set Intersection function and the frequency moments of a data stream, originating from Alon *et al.* (1999).

Remark 1.4. *Clearly, the* number of passes *is also a lower bound on the* running time *of a Stack Machine. If interpreted in this way, our method comes with an important limitation. Specifically, by using a reduction to communication complexity, it is not clear how to obtain any super-linear lower bounds, because the communication complexity of any function is at most linear. Still, as explained in* Section 7, *Stack Machines with logarithmic space and few passes over the input are quite powerful, so a lower bound on passes can be seen as interesting in its own right.*

Remark 1.5. *Some of the technical effort in our proofs is directed toward dealing with Stack Machines that have* two-way *rather than* one-way *access to their input tape. We point out that these restrictions are polynomially, but super-linearly, related: A Turing Machine (with or without a stack) that makes $r(N)$* two-way *passes over an input of size N can be simulated by a similar Turing Machine M' that makes $N \cdot r(N)$* one-way *passes. However, the method presented in this work can only derive lower bounds of the form $r(N) \geq \Omega\left(N^{\alpha}\right)$ for some $0 < \alpha \leq 1$. Therefore, we cannot use our current methods to first prove lower bounds for one-way* *access and then transfer them to two-way access using the simple argument above.*

1.2. Related work. Turing Machines with limited reversals have been studied before, e.g., Chen & Yap (1991). However, in that line of work, reversals are bounded on all tapes. By comparison, the Stack Machines we consider are significantly more powerful, because reversals are unbounded on both their space-bounded internal tapes and their stack.

This type of a reduction, connecting efficient communication protocols and space-bounded computation, is not new. One of the first examples is Babai *et al.* (1992), which derives time-space trade-offs for multi-head Turing Machines. Subsequently, such reductions have been used to derive lower bounds in streaming (Alon *et al.* 1999), an area of computer science whose object of study is the power of Turing Machines with small space and a single (or, very few) pass(es) over the input tape.

At the technical level, Stack Machines are related to (r, s, t) read-write stream algorithms. The latter are Turing Machines that have a constant number t of "external" read-write tapes, several "internal" tapes of combined space s, and a total number r of passes, counted over *all* external tapes. These machines were introduced by Grohe & Schweikardt (2005) as an extension of the standard streaming model, in which the machines have access to a single external read-only tape. Grohe & Schweikardt (2005), Grohe *et al.* (2006) derived several lower bounds for deterministic and one-sided error randomized read-write stream (r, s, t) algorithms. Beame *et al.* (2007) showed that two of the standard measures that are used to bound communication complexity, discrepancy and corruption, can be used to derive lower bounds for computing certain direct-sum type functions with two-sided error randomized stream algorithms. Finally, Beame & Huynh-Ngoc (2008) gave a simpler and more direct reduction between number-in-hand communication protocols and read-write stream algorithms, obtaining trade-off lower bounds between space and number of passes that inspired some the technical arguments in this paper.

Stack Machines and read-write stream algorithms are somewhat similar at the technical level because they both augment a space-bounded Turing Machine, with *an unbounded stack*, on the one hand, and with *several pass-bounded read-write tapes*, on the other hand. However, the motivation for considering these models is essentially different. Stack Machines are intimately connected to combinatorial circuits, which in turn model efficient parallel computation, whereas read-write stream algorithms model efficient computation in the presence of unlimited but slow "external" memory and fast but limited "internal" memory. In fact, we show that read-write stream algorithms cannot efficiently simulate Stack Machines, conditional on the widely believed complexity assumption that $\mathsf{E} \not\subseteq \mathsf{PSPACE}$.

It is the treatment of the stack that constitutes the main technical novelty in our result. To put this into perspective, observe that a Turing Machine equipped with *two* unbounded stacks can decide any decidable language, using no workspace at all and a single pass over the input (e.g., Sipser 1997, Problem 3.9). In comparison, Turing Machines with *any constant number* of external tapes with space bound s and pass bound r can only compute languages in $\mathsf{DSPACE}\,(r^2 \cdot s)$ (Hernich & Schweikardt 2008, Lemma 4.8).

1.3. Organization. The heart of our result is a new communication complexity reduction. In Section 2, we give an outline of this reduction in a simplified setting which still involves most difficulties. In Section 3, we give the formal definitions needed for our results. In Section 4, we give the proof of the main reduction, Theorem 1.1. In Section 5, we give the proofs for the consequences of our reduction (see Corollaries 1.2 and 1.3). In Section 6, we put the unconditional lower bounds for logarithmic space Stack Machines in perspective, showing that Stack Machines with a single pass over the input can compute languages outside NC (conditional on $\mathsf{PSPACE} \subsetneq \mathsf{EXP}$), and also that such Stack Machines cannot be simulated by the read-write stream algorithms of Grohe & Schweikardt (2005) (conditional on $\mathsf{E} \not\subseteq \mathsf{PSPACE}$). We conclude in Section 7.

2. Outline of the argument

Consider the regular Inner Product function $\mathrm{IP} = \mathrm{IP}_{2,n} : (\{0,1\}^n)^2 \to \{0,1\}$, which is defined by $\mathrm{IP}_{2,n}(x_1, x_2) = \sum_{i=1}^{n} x_{1,i} \cdot x_{2,i}$

where the operations are over $\mathbb{F}_2$. Let (x_1, x_2) be an input, with $x_p = (x_{p,1}, \ldots, x_{p,n})$, where $x_{p,j} \in \{0, 1\}$ for $p \in [2]$ and $j \in [n]$. In the communication complexity world, we are interested in computing this function with 2 player protocols, in which player p gets x_p, for $p \in [2]$. We know that $R_2(\mathrm{IP}_{2,n}) \geq \Omega(n)$ (Chor & Goldreich 1988), where R_2 denotes the private-coin, randomized 2-party communication complexity, where the error is a constant smaller than 1/2 (see e.g., Kushilevitz & Nisan 1997 for a detailed definition). In the TM world, we are interested in computing this function when its input is given on a tape in the natural way: first the n bits of x_1, then the n bits of x_2. The input size for the TM is $N = 2 \cdot n$. In this section, we give a simplified outline of our reduction, ignoring probabilistic machines and two-way passes. We study the trade-off between the space $s(N)$ and the number of *one-way passes* $r(N)$ that a *deterministic* TM needs. In the full proof, we allow $k \geq 2$ players, we allow *two-way passes*, and we allow *randomization* in the TM.

In the absence of a stack. When there is no stack, there is a simple, well-known, simulation of a space- and pass-bounded TM by a 2 player communication protocol. Player 1 simulates the TM until the head first crosses into the input zone containing x_2, at which point it sends the full state of the TM to player 2. The latter continues the simulation until the head crosses back into x_1 and so on. In total, the communication in this protocol is $\mathrm{O}(s(N) \cdot r(N))$. By the communication complexity lower bound, we get $s(N) \cdot r(N) \geq \Omega(n) = \Omega(N)$. This simple simulation breaks down in the case of a SM because there might be transfer of information via the stack between the parts of the computation when the input head is scanning x_1 and x_2. One way to see this is to observe that the configuration of a TM can be described by $\mathrm{O}(s)$ bits, whereas that of a SM might require up to $2^{\Theta(s)}$ bits.

A framework for dealing with a stack. In order to deal with the presence of a stack, we adapt a framework that was originally introduced by Beame & Huynh-Ngoc (2008) in the context of TMs with several external tapes.

Consider a function f which we know is hard in the communication model. We assume that an efficient SM M exists for a related function F. Using this assumption, we build an efficient communication protocol P for f. Let x be an input to f in P. The players in P first construct an input v to F (which contains x and some public "padding"), they simulate M on v, computing $F(v)$, and finally they derive $f(x)$ from $F(v)$. The simulation of M on v must be efficient so that communication lower bounds apply.

Let Γ be the sequence of configurations of M on v. In P, the players simulate Γ using a series of publicly and privately simulated sections. The public simulation is "cheap" in the sense that it does not cost any communication. The private simulation is "expensive" because, at the end of each private section, the player simulating it must communicate something in order for the other player(s) to "know where they are" in Γ. Each section in Γ where the input head scans a symbol from x (which is the input to P) is to be simulated privately by the player who knows that symbol. Moreover, the basic idea is that, since we do not have any bounds on how a SM can use its stack, we want to avoid communicating stack content in P. Thus, we do not mind if either: A symbol is pushed on the stack during public simulation and popped during either public or private simulation; or a symbol is pushed on the stack in a privately simulated section by player p and later popped in a (possibly different) privately simulated section by the same player p. What we want to avoid is the remaining scenario: A symbol is pushed on the stack in a privately simulated section by player p_1 and later popped in a privately simulated section by player $p_2 \neq p_1$. Informally, the way we achieve this "protection" against M using its stack on x is by "hiding" x into the larger input v, so that, with high probability, M only uses its stack for "meaningless" computation.

Concretely, let $m = n = \sqrt{N/2}$. Suppose that in a communication protocol two players want to compute $\mathrm{IP}_{2,n}$, and they have access to an efficient (space $s(N)$, one-way passes $r(N)$) SM for $\mathrm{IP}_{2,N/2}$. We think of $\mathrm{IP}_{2,N/2}$ as m instances of $\mathrm{IP}_{2,n}$ glued together by a top-level XOR_m gate.

> In the communication protocol P, the players get inputs $x_1, x_2 \in \{0,1\}^n$, respectively. They begin by choosing a random $i^* \in [m]$ and random $y_{1,i}, y_{2,i} \in \{0,1\}^n$ for $i \in [m] \setminus \{i^*\}$, all using public coins. Let $v = (v_1, v_2)$ be the input to M defined by, $v_{p,i^*} := x_p$ and $v_{p,i} := y_{p,i}$, for $i \neq i^*$ and $p \in [2]$. Henceforth, the goal of P is to simulate M on v.

Thus, both players know most of the input to the SM, except for two pieces of n bits each, where their respective inputs from the communication protocol are embedded in the N bit input v to M. Furthermore, observe that if they were indeed able to simulate the SM, they could compute the output for the protocol as $\mathrm{IP}(x_1, x_2) = \mathrm{IP}(v_{1,i^*}, v_{2,i^*}) = \mathrm{XOR}_2\left(\mathrm{IP}(v_1, v_2), \mathrm{XOR}_{m-1}\left(\mathrm{IP}(v_{1,i}, v_{2,i}) \mid i \neq i^*\right)\right)$.

Unique stack symbols. For the purposes of the analysis, we assume that each symbol on the stack is given a unique tag. Thus, even though the same symbol might appear many times on the stack, we assume we can distinguish between any of those appearances. In particular, to say that "a symbol is placed on the stack in configuration γ_1 and popped in configuration γ_2" formally means that the stack level is the same in γ_1 and in the configuration immediately following γ_2; and the stack level is strictly higher in any intermediate configuration between those two.

Corrupted instances. A key concept in our argument is that of a "corrupted instance". For a SM M and an input v, we say that *instance i is corrupted in v on M* if, during the run of M on v, a symbol is placed on the stack while the input head is scanning $v_{p_1,i}$ and popped while it is scanning $v_{p_2,i}$, where $p_1 \neq p_2$.

Three claims. Henceforth, our argument is built on the following three claims:

(i) The number of corrupted instances in any one input is *small.*
(ii) The choice of i^* and of v in the communication protocol is *statistically independent* of each other.

(iii) If the input x to the communication protocol *is not* embedded in a corrupted instance in the input v to the SM, then the protocol can *efficiently* simulate the SM.

The intuition for claim (i). Fix a SM M and an input v. We want to give a bound for the number of corrupted instances in v on M. We associate each corrupted instance $i \in [m]$ with a unique 4-tuple $(l_1 \leq l_2, p_1 \neq p_2)$, such that a symbol is pushed on the stack in the pass l_1 over $v_{p_1,i}$ and popped in the pass l_2 over $v_{p_2,i}$. Note, for each corrupted instance there might be several such 4-tuples to choose from, here, we associate i with only (any) one of them. We say that i *is corrupted* by this 4-tuple.

Let $l_1 \leq l_2 \leq r$ and let $p_1 \neq p_2 \in [2]$. During the pass l_1 over v_{p_1}, the input head scans the instances in v going from left-to-right in the order: $1, 2, \ldots, m$. The same is true for the pass l_2 over v_{p_2}. Assume instances $i_1 \neq i_2$ are corrupted by this particular 4-tuple. Without losing generality, say $1 \leq i_1 < i_2 \leq m$. We see that the stack symbol associated with i_1 is pushed first, then the stack symbol associated with i_2, then the stack symbol associated with i_1 is popped, and finally the stack symbol associated with i_2 is popped. This contradicts the First-In Last-Out access semantics of a stack. Hence, at most one instance is corrupted by any one 4-tuple.

Since the number of 4-tuples is at most $2 \cdot r^2$, we derive that at most $\mathrm{O}(r^2)$ instances can be corrupted in v on M. Thus, as long as $r^2/m = o(1)$, the fraction of corrupted instances in any one input is small. This argument is extended to work for *two-way* access, and slightly improved, as Lemma 4.1.

The intuition for claim (ii). To see why we need (ii), observe that in the communication protocol P, v is constructed *based on* i^*. For (i) to be useful, we would like the choice of i^* to probabilistically "hide" the instance containing the input to P from the set of corrupted instances in v on M. It seems that we would need i^* to be chosen *after* v.

To achieve (ii), we use an argument that goes through distributional communication complexity (see e.g., Kushilevitz & Nisan 1997): For every distribution D on the inputs to $\mathrm{IP}_{2,n}$ in the communication protocol, the players choose the $m-1$ "decoy" instances from that same distribution D, so that v is distributed according to D^m *independently of* i^*. In this case, we can permute the choices of v and i^*, so that claim (i) gives a bound on the probability the communication protocol embeds the real input in a corrupted instance. We go back to randomized communication complexity (for which we have lower bounds) using Yao's min–max principle (Kushilevitz & Nisan 1997, Theorem 3.20) connecting these two measures. This argument is made precise inside the proof of Theorem 1.1.

The intuition for claim (iii). Assume that the instance i^*, where the real input $x = (x_1, x_2)$ to the communication protocol P is embedded in v, is *not* corrupted. We argue that P can efficiently simulate M.

Let Γ be the sequence of configurations that M goes through on input v. We say that a configuration is *input-private to player* p if the input head is scanning a symbol from v_{p,i^*} (which is where x_p is embedded). Intuitively, we want player p to simulate the transition out of a configuration that is input-private to player p, because only it knows the symbols in $v_{p,i^*} = x_p$. Moreover, symbols might be pushed on the stack in such a transition, in which case we say that any such stack symbol is *private to player* p. We say that a configuration is *stack-private to player* p if the transition out of this configuration pops a stack symbol that is private to player p. Intuitively, we want player p to simulate such a transition in order to avoid communicating stack contents during the protocol. All other configurations are *public*. Since we assumed instance i^* is not corrupted, we know there is no configuration which is input-private and stack-private to different players.

The players simulate Γ by alternating between public and private simulation. At the end of each privately simulated section, the player performing that simulation communicates: the state of the SM, the lowest stack level reached during the private simulation section, the current stack level, and the top stack symbol. It can be shown that *the cost of the communication at the end of each privately simulated section is* $\mathrm{O}(s)$. In the proof of Lemma 4.2, we argue that the information communicated is sufficient for the players to obtain a "hollow view" of the stack: Each player knows the public stack symbols, the stack symbols which are private to itself, and the locations of the stack symbols private to other players. Moreover, this hollow view is sufficient for the players to continue the simulation until the end of Γ.

Finally, let us give an informal bound on the amount of communication. Observe that input-private configurations form exactly $2 \cdot r$ contiguous subsequences in Γ. Let us denote these contiguous zones of input-private configurations in Γ by S_a, for $1 \leq a \leq 2 \cdot r$. Let γ_a and γ'_a be the configurations at the beginning and end of S_a, respectively. Observe that the stack at the end of S_a (in γ'_a) contains at most *a contiguous zones of private stack symbols*, at most one such zone corresponding to each previous input-private section $S_{a'}$, with $a' \leq a$. Some of these zones might be accessed before S_{a+1}. As explained in detail in the proof of Lemma 4.2, the players in the protocol can simulate the sequence of configurations between γ'_a and γ_{a+1} using *as many privately simulated sections as there are contiguous zones of private stack symbols in* γ'_a. Since $a \leq 2 \cdot r$, the number of privately simulated sections between γ'_a and γ_{a+1} is $\mathrm{O}(r)$. Summing over all a and taking into account the privately simulated sections S_a, we get that, in total, the simulation of Γ can be performed using at most $\mathrm{O}(r^2)$ privately simulated sections. Hence, the communication bound for the entire protocol is $\mathrm{O}(r^2 \cdot s)$.

Putting (i), (ii) and (iii) together, assuming that we have a SM for $\mathrm{IP}_{2,N/2}$, we obtain a randomized communication protocol for $\mathrm{IP}_{2,n}$ that has error $\mathrm{O}(r^2/m)$ and cost $\mathrm{O}(r^2 \cdot s)$. As long as $r^2/m = o(1)$, the known randomized communication complexity lower bound $R_{2,\epsilon}(\mathrm{IP}_{2,n}) \geq \Omega(n)$ applies (Chor & Goldreich 1988), where $R_{2,\epsilon}$ denotes the 2-party randomized communication complexity for protocols with error $\leq \epsilon$ (see e.g., Kushilevitz & Nisan 1997 for definitions). Thus, we obtain the trade-off lower bound $r(N)^2 \cdot s(N) \geq \Omega(n) = \Omega\left(N^{1/2}\right)$.

What was left out of this outline. First, *the number of players* in this outline is set to $k = 2$. While this is sufficient to derive SM trade-off lower bounds for Inner Product-like functions, we need to allow the number of players to be a function of the input size, that is, $k = k(n)$, in order to obtain inapproximability results for the frequency moments problem in Corollary 1.3.

Second, the Inner Product function is of such a nature that no matter how the players in P choose the decoy instances $y_{p,i} \in \{0,1\}^n$ for $p \in [2]$ and $i \neq i^*$, they can always retrieve $f(x) = \mathrm{IP}(x_1, x_2)$ from $F(v) = \mathrm{IP}(v_1, v_2)$ by computing the XOR of all decoy instances $\mathrm{IP}(y_{1,i}, y_{2,i})$ together with $F(v)$. This is a property of the top-level gate in IP, which is XOR. In order to deal with the frequency moments problem, we use a reduction involving the promise Set Intersection problem, which has top-level gate OR. In this case, in order to be able to retrieve $f(x)$ from $F(v)$, the decoy instances *must* be 0-instances, and only then do we have $f(x) = F(v)$. To accommodate for this requirement and still achieve claim (ii), we need to treat 0-inputs and 1-inputs differently inside the proof of Theorem 1.1.

Third, we need to fill in the details of the simulation informally described as claim (iii).

Fourth, there is the issue of *two-way versus one-way passes*. This outline only considers how to obtain trade-off lower bounds for SMs with one-way access to the input. A SM with two-way access to the input can compute IP with only 2 passes (use the stack to store one vector for the inner product). The point where this framework breaks down is claim (i), which is outright false. Consider the case when $p_1 = 1$, $p_2 = 2$, the pass $l_1 = 1$ is left-to-right, and the pass $l_2 = 2$ is right-to-left. We see that the instances are visited in the pass 1 in the order $1, 2, \dots, m-1, m$, and they are visited in the pass 2 in the order $m, m-1, \dots, 2, 1$. Then, the associated "corrupting" stack symbols can be pushed in the order $1, \dots, m$ and popped in the order $m, \dots, 1$, without violating the access semantics of a stack. Thus, potentially *all* instances can be corrupted by a single 4-tuple $(1, 2, 1, 2)$, which is precisely what happens in the simple 2-pass protocol seen earlier.

In order to obtain trade-off lower bounds for SMs with two-way access, we need to fix claim (i). To that end, we "scramble" the order in which the m instances are seen in pass l_1 over v_{p_1} and in pass l_2 over v_{p_2}, no matter what p_1 and p_2 are, and no matter whether the passes l_1 and l_2 are left-to-right or right-to-left. Informally, we achieve this by reordering the m instances corresponding to each player using a family of permutations $\Phi = (\varphi_1, \ldots, \varphi_k)$, with one permutation for each player that has the following property.

> For $p \in [k]$, let $\sigma_p := (\varphi_p(1), \ldots, \varphi_p(m))$ and $\sigma_p^{\text{rev}} := (\varphi_p(m), \ldots, \varphi_p(1))$. Then, for every $p_1 \neq p_2$, $\sigma_1' \in \{\sigma_{p_1}, \sigma_{p_1}^{\text{rev}}\}$, and $\sigma_2' \in \{\sigma_{p_2}, \sigma_{p_2}^{\text{rev}}\}$, σ_1' and σ_2' have *as small a common subsequence as possible.*

It turns out that there exist families Φ where any such common subsequence has size $\mathrm{O}(\sqrt{m})$ (Beame & Huynh-Ngoc 2008).

3. Definitions and facts

We use standard definitions for computational complexity classes (see, e.g., Arora & Barak 2009) and communication complexity notions (see, e.g., Kushilevitz & Nisan 1997).

Notation. We use n to denote the size of the input to a player in a communication protocol. The number of players is $k = k(n)$, which is a non-decreasing function of n. A (randomized) k-player number-in-hand protocol (NIH) is defined as a communication protocol where each player uses private randomness holds only her own part ($1/k$-th) of the input and communication happens through a shared blackboard (for a detailed definition see Kushilevitz & Nisan 1997). We use $f = f_{k,n} : (\{0,1\}^n)^k \to \{0,1\}$ to denote a generic boolean function which we are interested in computing in the communication model.

We use F to denote a generic function that we are interested in computing in the Stack Machine model. In general, F consists of m instances of a base function $f_{k,n}$ combined together by combining function $h_m : \{0,1\}^m \to \{0,1\}$. The number of instances, $m = m(n)$, is a non-decreasing function of n. We denote by N the size of the input to F, so $N = N(n) = k \cdot m \cdot n$. We use $s = s(N)$ and $r = r(N)$ to denote non-decreasing functions bounding the space and the number of passes a SM is allowed (these are defined below).

The Inner Product function $\mathrm{IP}_{2,n} : (\{0,1\}^n)^2 \to \{0,1\}$ is defined as in the beginning of the previous section. The Set Intersection function $\mathrm{SetInt}_{k,n} : (\{0,1\}^n)^k \to \{0,1\}$ is defined by $\mathrm{SetInt}_{k,n}(x_1, \ldots, x_k) = \bigvee_{i=1}^n \left(\wedge_{j=1}^k x_{j,i}\right)$. In the *promise* Set Intersection function $\mathrm{pSetInt}_{k,n}$, we are promised that the inputs satisfy the following: There is at most one $i \in [n]$ such that $\wedge_{j=1}^k x_{j,i} = 1$; and for every $i \in [n]$ where $\wedge_{j=1}^k x_{j,i} = 0$, there is at most one $j \in [k]$ such that $x_{j,i} = 1$.

Stack machines. A deterministic (uniform) *Stack Machine* (SM) is a Turing Machine with a fixed number of internal tapes, a single read-only external tape of unbounded size containing the input and a *stack*. The stack is implemented as an additional one-way infinite tape that follows the access semantics of a push-down store (Cook 1971). A randomized SM is allowed to flip an unbiased coin before every single transition. We say that a randomized SM computes a (boolean) function with error δ if it computes correctly (two-sided) with probability $\geq 1 - \delta$.

Let s, r be non-decreasing functions. A SM has a *space bound s* if, for every N, the *size of its work tapes* is at most $s(N)$ on inputs of length N. A SM has a *pass bound r* if, for every N, the *number of passes it makes over the input tape* is at most $r(N)$ on inputs of length N. Neither of these bounds refers in any way to the stack.

Without losing generality, we make the following assumptions about SMs:

(1) A SM has enough space on its work tapes to record the position of the head on the input tape. That is, $s(n) \geq \Omega(\log N)$.

(2) In every pass over the input tape, the input head travels from one end of the tape to the other.

(3) The stack in a SM is empty both at the beginning and at the end of its computation.

(4) In every transition of a SM, *exactly one* of the following occurs:

- the input head moves on the input tape, referred to as a *move transition*; or
- a symbol is pushed on the stack, referred to as a *push transition*; or

  - a symbol is popped from the stack, referred to as a *pop transition.*

It turns out that we need to consider *non-uniform* Stack Machines in one of our proofs. Naturally, the non-uniform model is more powerful than the uniform one, so lower bounds for the non-uniform model are stronger. Moreover, communication protocols are by definition a non-uniform model of computation, so when a communication protocol simulates a Stack Machine, we might as well assume the Stack Machine is also non-uniform.

For a fixed input length N, a *stack automaton* A_N is an automaton with a read-only input tape of size N, a set of states whose size can now depend on N, and a stack. Assumptions (1)–(4) still apply. A stack automaton A_N has a *space bound* $s(N)$ if A_N has at most $2^{s(N)}$ states. A_N has a *pass bound* $r(N)$ if A_N makes at most $r(N)$ passes over its input.

A *non-uniform Stack Machine* is a collection $A = (A_N)_{N\in\mathbb{N}}$ of stack automata, one for each input length. For non-decreasing functions r, s, a non-uniform SM $A = (A_N)$ has a space bound s and a pass bound r if, for every N, A_N has a space bound $s(N)$ and a pass bound $r(N)$, respectively.

A *(full) configuration* of a SM contains:

- the state and the full contents of the work tapes in the uniform case, or simply the state in the non-uniform case;
- the location of the head on the input tape; and
- the entire contents of the stack.

A *surface configuration* of a SM is the same as a full configuration, but instead of the entire stack contents, it only includes

- the top stack symbol.

FACT 3.1. *Let $A = (A_N)$ be a non-uniform SM that halts on every input and runs in space $s = s(N)$. Then, for large enough N and for every input w of length N, the maximum stack height of A during its run on w is at most $2^{4\cdot s(N)}$.*

PROOF (Proof of Fact 3.1). Assume the claim does not hold. Let c denote the (constant) number of symbols in the stack alphabet of A. Let N be large enough so that $s(N) \geq \log c$ (note that $s(N) \geq \log N$ by assumption (1)). Fix an input w of size N on which the stack reaches height more than $2^{4 \cdot s(N)}$. Let Γ be the sequence of configurations of M on w. Let $\gamma \in \Gamma$ be a configuration in which the stack contains strictly more than $2^{4 \cdot s(N)}$ symbols.

For every stack level l, with $1 \leq l \leq 2^{3 \cdot s(N)} + 1 < 2^{4 \cdot s(N)}$, let γ_l be the last configuration preceding γ in which the symbol at level l is at the top of the stack. Obviously, the transition out of γ_l is a push transition. For every such l, let $\bar{\gamma}_l$ denotes the surface configuration corresponding to γ_l.

Observe that a surface configuration can be represented using $\log N + s(N) + \log c \leq 3 \cdot s(N)$ bits. Since there are more than $2^{3 \cdot s(N)}$ symbols on the stack, a surface configuration must be repeated. Accordingly, let $1 \leq l_1 < l_2 \leq 2^{3 \cdot s(N)} + 1$ be stack levels such that $\bar{\gamma}_{l_1} = \bar{\gamma}_{l_2}$.

By definition, γ_{l_1} is the last configuration when the stack level was l_1. Hence, the SM got from γ_{l_1} to γ_{l_2} without accessing a symbol at level l_1 or below. But then, since $\bar{\gamma}_{l_2} = \bar{\gamma}_{l_1}$, the SM is bound to repeat this part of its computation, eventually ending up in a configuration γ_{l_3} with $l_3 - l_2 = l_2 - l_1$ and $\bar{\gamma}_{l_3} = \bar{\gamma}_{l_2}$. This process is repeated indefinitely, so the SM does not halt on w, a contradiction. □

Permutations, sequences, and sortedness. Let S_m denotes the set of all permutations of $[m]$. Let id denotes the identity permutation. For a permutation $\pi \in S_m$, let $\text{seq}(\pi)$ denotes the m-element sequence $(\pi(1), \pi(2), \ldots, \pi(m))$. For a sequence σ, let σ^{rev} denotes σ reversed. For example, $\text{seq}(\pi)^{\text{rev}} = (\pi(m), \ldots, \pi(1))$. For two sequences σ_1, σ_2, let $\text{LCS}(\sigma_1, \sigma_2)$ denote their *longest common* (not necessarily contiguous) *subsequence.* For a sequence σ, let $|\sigma|$ denotes its length.

Consider two permutations $\varphi_1, \varphi_2 \in S_m$. We measure their *relative sortedness* according to the following experiment.

> Intuitively, imagine we lay out $\text{seq}(\varphi_1) = (\varphi_1(1), \ldots, \varphi_1(m))$ on one tape and $\text{seq}(\varphi_2) = (\varphi_2(1), \ldots, \varphi_2(m))$ on another tape. We are allowed two passes, one over each tape, in parallel, with the heads moving

> independently of each other. We can choose to make the pass over each tape left-to-right or right-to-left. We "eliminate" an index $a \in [m]$ if, at some point during the two passes, a is the value being scanned on both tapes. We define the relative sortedness of φ_1 and φ_2 to be the maximum number of indices we can eliminate in this way.

Formally,

$$\begin{aligned}&\operatorname{relsort}(\varphi_1, \varphi_2)\\ &\quad := \max\left\{ |\mathrm{LCS}(\sigma_1, \sigma_2)| \;\middle|\; \forall i \in [2], \sigma_i \in \{\mathrm{seq}(\varphi_i), \mathrm{seq}(\varphi_i)^{\mathrm{rev}}\}\right\}\end{aligned}$$

For example, for

$$\varphi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \underline{2} & \underline{1} & \underline{4} & \underline{3} & 5 & 6 \end{pmatrix} \quad \text{and} \quad \varphi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \underline{3} & 6 & \underline{4} & \underline{1} & 5 & \underline{2} \end{pmatrix},$$

we have $\operatorname{relsort}(\varphi_1, \varphi_2) = 4$, because $(2, 1, 4, 3)$ is a common subsequence of $\mathrm{seq}(\varphi_1)$ and $\mathrm{seq}(\varphi_2)^{\mathrm{rev}}$.

For a set of permutations $\Phi = \{\varphi_1, \dots, \varphi_t\}$, we define its relative sortedness to be the maximum relative sortedness of any two of its members. Formally,

$$\operatorname{relsort}(\Phi) := \max\left\{ \operatorname{relsort}(\varphi_i, \varphi_j) \;\middle|\; i \neq j \in [t]\right\}.$$

We use the following Fact to obtain sets of permutations with small relative sortedness.

Fact 3.2. *Let $\varphi_1, \varphi_2 \in S_m$. If* $\operatorname{relsort}(\varphi_1, \varphi_2) = t$, *then* $\mathrm{seq}(\varphi_2^{-1} \circ \varphi_1)$ *contains a monotone subsequence of length t.*

Proof (of Fact 3.2). First, assume $(i_1, i_2, \dots, i_t)$ is a common subsequence of $\mathrm{seq}(\varphi_1)$ and $\mathrm{seq}(\varphi_2)$. In this case, the corresponding sequences of pre-images are both non-decreasing, $(\varphi_1^{-1}(i_1) < \varphi_1^{-1}(i_2) < \cdots < \varphi_1^{-1}(i_t))$ and $(\varphi_2^{-1}(i_1) < \varphi_2^{-1}(i_2) < \cdots < \varphi_2^{-1}(i_t))$. Let $i'_j := \varphi_1^{-1}(i_j)$ for $j \in [t]$, and observe that $\varphi_2^{-1}(\varphi_1(i'_j)) = \varphi_2^{-1}(i_j)$. Then, $(i'_1, i'_2, \dots, i'_t)$ is a subsequence of $(1, 2, \dots, m)$, so $(\varphi_2^{-1} \circ \varphi_1(i'_1) < \varphi_2^{-1} \circ \varphi_1(i'_2) < \cdots < \varphi_2^{-1} \circ \varphi_1(i'_t))$ is an increasing subsequence of $(\varphi_2^{-1} \circ \varphi_1(1), \varphi_2^{-1} \circ \varphi_1(2), \dots, \varphi_2^{-1} \circ \varphi_1(m)) = \mathrm{seq}(\varphi_2^{-1} \circ \varphi_1)$.

The cases when the subsequence $(i_1, i_2, \ldots, i_t)$ comes from $\mathrm{seq}(\varphi_1)^{\mathrm{rev}}$ instead of $\mathrm{seq}(\varphi_1)$, or from $\mathrm{seq}(\varphi_2)^{\mathrm{rev}}$ instead of $\mathrm{seq}(\varphi_2)$, or both, are treated similarly. The only difference is that one or both of the corresponding sequences of pre-images are now decreasing, which might change the monotonicity of the resulting subsequence of $\mathrm{seq}(\varphi_2^{-1} \circ \varphi_1)$. □

By Fact 3.2, to obtain permutations with small relative sortedness, it is enough to require that the (sequences of their) corresponding compositions have small monotone subsequences. This method comes with a limitation, as by the Erdös–Szekeres Theorem (Erdös & Szekeres 1935), every sequence of m distinct elements contains a monotone subsequence of length $\lceil \sqrt{m} \rceil$. Thus, the smallest relative sortedness that we can achieve between two permutations using Fact 3.2 is $\Theta(\sqrt{m})$. Grohe & Schweikardt (2005) show that, in fact, a very simple permutation has asymptotically optimal relative sortedness with the identity permutation.

Fact 3.3 (Lemma 6 in Grohe & Schweikardt 2005). *For every m, there exists a permutation $\psi_m^* \in S_m$ such that* $\mathrm{relsort}(\mathrm{id}, \psi_m^*) \leq 2 \cdot \sqrt{m} - 1$. *Furthermore, ψ_m^* can be computed in space* $\mathrm{O}(\log m)$.

Beame & Huynh-Ngoc (2008) use a simple counting argument to show that a similar bound, still asymptotically optimal, can be achieved even for sets of permutations.

Fact 3.4 (Corollary 2.2 in Beame & Huynh-Ngoc 2008). *Let $k = k(m)$ be a function such that $k \leq m^{\mathrm{O}(1)}$. There exists a family $\Phi^* = \left(\Phi^*_{k,m}\right)_m$ where $\Phi^*_{k,m} = \{\varphi_1, \ldots, \varphi_k\}$ is a set of k permutations from S_m, such that* $\mathrm{relsort}\left(\Phi^*_{k,m}\right) \leq \mathrm{O}(\sqrt{m})$.

Permuted functions. Let $f = f_{k,n} : (\{0,1\}^n)^k \to \{0,1\}$ be a *base function.* Let $h = h_m : \{0,1\}^m \to \{0,1\}$ be a *combining function.* Let $\Phi = \Phi_{k,m} = \{\varphi_1, \ldots, \varphi_k\}$ be a *set of k permutations* from S_m. In what follows, we define $\mathrm{LiftMix}(f_{k,n}, h_m, \Phi_{k,m}) : (\{0,1\}^{m \cdot n})^k \to \{0,1\}$ which is a *lift-and-mix* function that consists of m *instances of f, permuted by Φ, and combined by h.*

Let $v \in (\{0,1\}^{m \cdot n})^k$ be an input. Let $v = (v_1, \ldots, v_k)$, where $v_p \in \{0,1\}^{m \cdot n}$ for every $p \in [k]$. Let $v_p = (v_{p,1}, \ldots, v_{p,m})$, where

$v_{p,i} \in \{0,1\}^n$ for every $(p,i) \in [k] \times [m]$. Let $v_{p,i} = (v_{p,i,1}, \dots, v_{p,i,n})$, where $v_{p,i,j} \in \{0,1\}$ for every $(p,i,j) \in [k] \times [m] \times [n]$. When v is given as an input to a TM, v appears on the input tape as

$$\underbrace{v_{1,1}, \dots, v_{1,m},}_{v_1} \underbrace{v_{2,1}, \dots, v_{2,m},}_{v_2} \dots, \underbrace{v_{k,1}, \dots, v_{k,m},}_{v_k}$$

and for $(p,i) \in [k] \times [m]$, $v_{p,i}$ appears as the sequence of bits

$$v_{p,i,1}, v_{p,i,2}, \dots, v_{p,i,n}.$$

For $(p,i') \in [k] \times [m]$, we say that $v_{p,\varphi_p^{-1}(i')}$ is *the* i'*-th instance* (of f) *inside* v_p. For $i' \in [m]$, we define *the* i'*-th instance* (of f) *inside* v to be

$$v_{[i'],\Phi} := \left(v_{1,\varphi_1^{-1}(i')}, v_{2,\varphi_2^{-1}(i')}, \dots, v_{k,\varphi_k^{-1}(i')}\right) \in \left(\{0,1\}^n\right)^k.$$

For $(p,i) \in [k] \times [m]$, $v_{p,i}$ appears in $v_{[i'],\Phi}$ if and only if $i = \varphi_p^{-1}(i')$. Since this is equivalent to $i' = \varphi_p(i)$, we observe the following basic fact.

FACT 3.5. *The order on the tape of the* m *instances inside* v_p *is* $(\varphi_p(1), \varphi_p(2), \dots, \varphi_p(m))$*, which is precisely* $\mathrm{seq}(\varphi_p)$*. Thus, a left-to-right tape scan over* v_p *visits the* m *instances in the order* $\mathrm{seq}(\varphi_p)$*, and a right-to-left tape scan visits them in the order* $\mathrm{seq}(\varphi_p)^{\mathrm{rev}}$.

We now define the lift-and-mix function $\mathrm{LiftMix}(f,h,\Phi) : \left(\{0,1\}^{m\cdot n}\right)^k \to \{0,1\}$ by

$$\mathrm{LiftMix}(f,h,\Phi)(v) := h(f(v_{[1],\Phi}), f(v_{[2],\Phi}), \dots, f(v_{[m],\Phi})).$$

We embed an input to f into a specific instance of an input to F as follows. Let $x \in \left(\{0,1\}^n\right)^k$ be an input to f. Let $i^* \in [m]$ be an instance number. Let $\bar{y}_{-i^*} = (y_1, \dots, y_{i^*-1}, y_{i^*+1}, \dots, y_m)$ be a set of $m-1$ inputs to f, where $y_i \in \left(\{0,1\}^n\right)^k$ for $i \neq i^*$. We define $v_\Phi(i^*, x, \bar{y}_{-i^*})$ to be the input to F in which x is embedded at instance i^* and y_i is embedded at instance i, for $i \neq i^*$. Formally,

$$v_\Phi(i^*, x, \bar{y}_{-i^*}) := v \in \left(\{0,1\}^{m\cdot n}\right)^k \text{ such that } \begin{cases} v_{[i],\Phi} = x, & i = i^*, \\ v_{[i],\Phi} = y_i, & i \neq i^*. \end{cases}$$

Corrupted instances. Let M be a deterministic SM and v be an input of size $N = k \cdot m \cdot n$. We say that *instance* $i \in [m]$ *is corrupted in* v *on* M if there exist players $p_1 \neq p_2 \in [k]$ such that, during the run of M on v, a symbol is pushed on the stack when the input head is scanning $v_{p_1, \varphi_{p_1}^{-1}(i)}$ and that symbol is popped when the input head is scanning $v_{p_2, \varphi_{p_2}^{-1}(i)}$, where $\varphi_{p_1}, \varphi_{p_2} \in \Phi_{k,m}$. Observe that both strings above are part of $v_{[i],\Phi}$, which is instance i inside v.

Let $\mathrm{BAD}(M, v) \subseteq [m]$ be the set of all instances which are corrupted in v on M. Observe that this definition implicitly depends on the values of k, m, n, and Φ, so we should formally write $\mathrm{BAD}_{k,m,n,\Phi}(M, v)$. We use the shorter form for brevity.

Neutral element. We say that a combining function h *has a neutral element* $e \in \{0, 1\}$ if for every $b \in \{0, 1\}$ and for every $i \in [m]$,

$$h(\underbrace{e, \ldots, e}_{i-1}, b, \underbrace{e, \ldots, e}_{m-i}) = b.$$

Observe that both OR and XOR have neutral element $e = 0$.

4. The reduction

Theorem 1.1 (Restated). *Let $k = k(n) \geq 2$ be a non-decreasing function, and let $m = m(n) \geq 1$ be a non-decreasing function such that $k \leq m^{\mathrm{O}(1)}$. Let $N = N(n) := k \cdot m \cdot n$. Let $s = s(N)$ and $r = r(N)$ be non-decreasing functions. Let $\delta < 1/2$ be a constant. Let $\Phi = (\Phi_{k,m})_m$ be a family of permutations, where $\Phi_{k,m} = \{\varphi_1, \ldots, \varphi_k\}$ are k permutations from S_m. Let $d := k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Phi_{k,m}\right)/m$.*

Let $f = f_{k,n}$ be a boolean base function and let $h = h_m$ be a combining function with a neutral element. Let $F = F_N := \mathrm{LiftMix}\left(f_{k,n}, h_m, \Phi_{k,m}\right)$.

Assume there exists a randomized non-uniform SM M for F with space bound s, pass bound r, and error δ. Then, there exists a randomized k-player NIH communication protocol P for f, with cost $\mathrm{O}\left(k \cdot r \cdot s \cdot \log(k \cdot r)\right)$ and error at most $\delta + \mathrm{O}\left(d\right)$.

To prove this Theorem, we use the following two Lemmas, which correspond to claims (i) and (iii) in the outline from Section 2. Throughout this entire section, let n, k, m, N, s, r, Φ be as in Theorem 1.1. Furthermore, in what follows for the fixed Φ the players construct an input $v_\Phi(\cdot)$, which we denote by $v(\cdot)$ for simplicity.

The first Lemma gives a bound on the number of corrupted instances inside a fixed input. As such, it directly corresponds to claim (i) from the outline in Section 2. As opposed to that claim, the Lemma allows for a growing number of players $k = k(n)$ and for two-way passes over the input tape thanks to the "scrambling" of instances according to the family of permutations Φ.

LEMMA 4.1. *Let M' be a deterministic non-uniform SM with space bound s and (two-way) pass bound r. Let v' be an input of size N. Then, $|\mathrm{BAD}(M', v')| \leq \mathrm{O}\left(k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Phi_{k,m}\right)\right)$.*

The next Lemma corresponds to claim (iii) from the outline in Section 2. It says that if the players in a communication protocol embed their input in a non-corrupted instance, then the protocol can efficiently simulate the SM.

LEMMA 4.2. *Let M' be a deterministic non-uniform SM. Let $i^* \in [m]$ be an instance number, and let $\bar{y}_{-i^*} = (y_1, \ldots, y_{i^*-1}, y_{i^*+1}, \ldots, y_m)$ be a set of $m-1$ inputs to f. Then, there exists a deterministic k-player NIH communication protocol $P' = P'(M', i^*, \bar{y}_{-i^*})$ such that, on input x:*

- *if $i^* \in \mathrm{BAD}(M', v(i^*, x, \bar{y}_{-i^*}))$, then P' outputs "fail";*
- *otherwise, P' correctly simulates M' and outputs $M'(v(i^*, x, \bar{y}_{-i^*}))$;*
- *the cost of P' is $\mathrm{O}\left(k \cdot r \cdot \log(k \cdot r) \cdot s\right)$.*

We give the proofs of Theorem 1.1 and Lemmas 4.1 and 4.2 in the following three sections.

4.1. Proof of the main theorem. In this section, we prove Theorem 1.1 using Lemmas 4.1 and 4.2.

For every q, let M_q denote the deterministic non-uniform SM obtained by running M with random string q.

Let D be any distribution on the space $(\{0,1\}^n)^k$ of inputs to f. Below, we give a randomized protocol P_D for f with cost and error as described in Theorem 1.1, but with error computed over the choice of the input x from D *and* of the random coins ρ in P. By standard arguments,[1] we can fix ρ to obtain a deterministic protocol with the same error, but only over the choice of x from D. Since such a protocol exists for every distribution D, by Yao's min–max principle (e.g., Theorem 3.20 in Kushilevitz & Nisan 1997), the conclusion of Theorem 1.1 follows.

Without loss of generality, assume that the neutral element of the combining function h is $e = 0$. If instead we have neutral element $e = 1$, the same proof applies with the roles of 0 and 1 reversed.

If D has support only on 0-inputs or only on 1-inputs, P_D is trivial. Otherwise, let D_0 be the distribution D, conditioned on $f(x) = 0$. Similarly, we define D_1 as D conditioned on $f(x) = 1$.

Consider the following protocol P_D:

```
On input $x$ (where player $p$ gets $x_p$) and shared random string $\rho$:
    Publicly draw $i^*$ uniformly from $[m]$
    Publicly draw $\bar{y}_{-i^*} = (y_1, \ldots, y_{i^*-1}, y_{i^*+1}, \ldots, y_{m-1})$ from $(D_0)^{m-1}$
    Publicly draw $q$ uniformly
        (where $q$ has as many bits as the running time of $M$)
    Run $P' = P'(M_q, i^*, \bar{y}_{-i^*})$ (from Lemma 4.2) on input $x$, simulating $M_q$ on $v(x, i^*, \bar{y}_{-i^*})$
    If $P'$ outputs "fail", then $P_D$ outputs 1.
    Else, $P_D$ outputs the same value as $P'$, that is, $M_q(v(x, i^*, \bar{y}_{-i^*}))$
```

Observe that, since the neutral element of h is $e = 0$, and since the instances in $\bar{y}_{-i^*}$ are drawn from D_0, we always have $f(x) = F(v)$, where $v = v(i^*, x, \bar{y}_{-i^*})$. Clearly, the cost of P_D is

---

[1] That is, some ρ must perform well on the average. Now consider the deterministic algorithm that fixes the randomness to this ρ (see Theorem 3.20 Kushilevitz & Nisan 1997) for a detailed similar argument.

equal to the cost of P', which is $\mathrm{O}\left(k \cdot r \cdot \log(k \cdot r) \cdot s\right)$. To argue about the error in P_D, we define the following events:

$A = A(x, \rho)$: P_D is correct on input x with coins ρ,
$B = B(q, v)$: M is correct on input v with coins q (i.e., $M_q(v) = F(v)$),
$C = C(i^*, q, v)$: instance i^* is *not* corrupted in v on M_q (i.e., $i^* \notin \mathrm{BAD}(M_q, v)$).

Our goal is to show $\Pr\left[\overline{A}\right] \leq \delta + \mathrm{O}\left(d\right)$. By Lemma 4.2, the simulation in P' fails if and only if $\overline{C}$. Let $\alpha := \Pr_x[f(x) = 1]$. We write

$$\Pr[A] = \alpha \cdot \Pr[A|f(x) = 1] + (1 - \alpha) \cdot \Pr[A|f(x) = 0].$$

For the left term, consider the conditioning $f(x) = 1$. P_D is correct if either the simulation in P' fails (and P_D correctly outputs 1), or the simulation in P' does not fail and M_q is correct. Thus, $\overline{C} \vee (C \wedge B) \Rightarrow A$, in particular $B \Rightarrow A$, so $\Pr[A|f(x) = 1] \geq \Pr[B|f(x) = 1]$. The event $f(x) = 1$ is independent of the choice of q, and by the correctness condition of M, $\forall v, \Pr_q[B(q, v)] \geq 1 - \delta$. Hence, $\Pr[B|f(x) = 1] = \sum_x \Pr_{D_1}[\text{sample } x] \cdot \Pr[B|x] \geq \sum_x \Pr_{D_1}[\text{sample } x] \cdot (1 - \delta) = 1 - \delta$.

For the right term, consider the conditioning $f(x) = 0$. Then, P_D is correct whenever the simulation in P' works and M_q is correct, thus $B \wedge C \Rightarrow A$, and

$$\Pr\left[A \mid f(x) = 0\right] \geq \Pr\left[B \mid f(x) = 0\right] \cdot \Pr\left[C \mid B, f(x) = 0\right].$$

As before, $\Pr[B|f(x) = 0] \geq 1 - \delta$. Next, i^* is clearly statistically independent of x and q. Furthermore, under the conditioning $f(x) = 0$, we claim that i^* *is statistically independent from* v. To see this, notice how the distribution obtained on pairs (i^*, v) in P_D is the same as independently choosing m 0-inputs from D_0 and combining them in v, and choosing i^* uniformly from $[m]$. Thus, in the expression $\Pr[C(i^*, q, v)|B(q, v), f(x) = 0]$, the conditioning depends on (x, v, q), C itself depends on i^*, and i^* is statistically independent from (x, v, q).

By Lemma 4.1,

$$\forall (q,v), \quad \Pr_{i^*}\left[\overline{C(i^*,q,v)}\right] \le \frac{\mathrm{O}\left(k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Phi_{k,m}\right)\right)}{m} \le \mathrm{O}\left(d\right).$$

Then, $\Pr[C|B, f(x) = 0] > \sum_x \Pr_{D_0}(x) \cdot (1 - O(d)) = 1 - \mathrm{O}\left(d\right)$.

Putting everything together, we get

$$\begin{aligned} \Pr\left[A\right] &= \alpha \cdot \Pr\left[A|f(x) = 1\right] + (1-\alpha) \cdot \Pr\left[A|f(x) = 0\right] \\ &\ge \alpha \cdot (1-\delta) + (1-\alpha) \cdot (1-\delta) \cdot (1 - \mathrm{O}\left(d\right)) \\ &\ge (1-\delta) \cdot (1 - \mathrm{O}\left(d\right)). \end{aligned}$$

Then, $\Pr\left[\overline{A}\right] \le \delta + \mathrm{O}\left(d\right)$.

4.2. Bounding the number of corrupted instances. In this Section, we formally prove Lemma 4.1, which corresponds to claim (i) in the outline from Section 2.

Let M' be a deterministic non-uniform SM with space bound s and pass bound r. Let v' be an input of size N. We want to show that the number of corrupted instances in v' on M' (that is, $|\mathrm{BAD}(M',v')|$) is at most $\mathrm{O}\left(k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Phi_{k,m}\right)\right)$.

We say that *instance* $i \in \mathrm{BAD}(M',v')$ *is corrupted by the 4-tuple* (l_1, l_2, p_1, p_2) if $l_1 \le l_2$, $p_1 \ne p_2$, and this is the lexicographically smallest 4-tuple such that a symbol is pushed on the stack during pass l_1 with the input head scanning $v_{p_1,\varphi_{p_1}^{-1}(i)}$ (which is instance i inside v_{p_1}), and popped from the stack during pass l_2 with the input head scanning $v_{p_2,\varphi_{p_2}^{-1}(i)}$ (which is instance i inside v_{p_2}). In what follows, we first consider the question of how many instances can be corrupted by one 4-tuple. Then, we argue that not all 4-tuples can simultaneously corrupt instances.

Fix one 4-tuple (l_1, l_2, p_1, p_2). Let $I \subseteq [m]$ be the set of instances that are corrupted by this 4-tuple and let $t = |I|$. Label the instances in I by $(i_1, i_2, \ldots, i_t)$, in the order in which they are visited in the pass l_1 over v_{p_1}. By Fact 3.5, we see that, depending on whether the pass l_1 is left-to-right or right-to-left, $(i_1, i_2, \ldots, i_t)$ is a subsequence of either $\mathrm{seq}(\varphi_{p_1})$ or $\mathrm{seq}(\varphi_{p_1})^{\mathrm{rev}}$. Furthermore, because of the First-In Last-Out semantics of a stack, the instances in I must be visited in the order $(i_t, \ldots, i_2, i_1)$ in the pass l_2 over v_{p_2}. Then, again by Fact 3.5, $(i_t, \dots, i_2, i_1)$ is a subsequence of either $\mathrm{seq}(\varphi_{p_2})$ or $\mathrm{seq}(\varphi_{p_2})^{\mathrm{rev}}$. Therefore, we see that in any case $(i_1, i_2, \dots, i_t)$ is a *common subsequence* of one of $\{\mathrm{seq}(\varphi_{p_1}), \mathrm{seq}(\varphi_{p_1})^{\mathrm{rev}}\}$ and one of $\{\mathrm{seq}(\varphi_{p_2}), \mathrm{seq}(\varphi_{p_2})^{\mathrm{rev}}\}$. Hence, $t \leq \mathrm{relsort}\,(\varphi_{p_1}, \varphi_{p_2}) \leq \mathrm{relsort}\,(\Phi_{k,m})$.

Next, consider the $r \times r$ matrix A, where $A[l_1, l_2] := 1$ if there exist $p_1, p_2 \in [k]$ such that some instance is corrupted by the tuple (l_1, l_2, p_1, p_2) and $A[l_1, l_2] := 0$ otherwise. Note that $A = 0$ under the main diagonal, because for a tuple to corrupt an instance we must have $l_1 \leq l_2$.

Moreover, for $j \geq 1$, consider the diagonal $l_2 - l_1 = j$, and two entries on this diagonal that are j' cells apart, for $1 \leq j' < j$. We claim that we cannot have $A[l_1, l_1 + j] = 1$ and $A[l_1 + j', l_1 + j + j'] = 1$. Assume this were true. Then, some instance i is corrupted because a symbol is pushed on the stack in the pass l_1 and popped in the pass $l_1 + j$, and another instance i' is corrupted because a symbol is pushed on the stack in the pass $l_1 + j'$ and popped in the pass $l_1 + j + j'$. But, with these settings, $l_1 < l_1 + j' < l_1 + j < l_1 + j + j'$, which contradicts the First-In Last-Out semantics of a stack.

Hence, for $j \geq 1$, the diagonal $l_2 - l_1 = j$ can contain at most r/j 1's. In total, we see that A contains at most $\mathrm{O}\,(r \cdot \log r)$ 1's.

Finally, for fixed $l_1 \leq l_2$, consider the $k \times k$ matrix $B = B_{(l_1, l_2)}$, where $B[p_1, p_2] := 1$ if some instance is corrupted by the tuple (l_1, l_2, p_1, p_2) and $B[p_1, p_2] := 0$ otherwise. First, consider the case when both passes l_1 and l_2 are left-to-right.

We claim that, for $j' \geq 1$, we cannot have $B[p_1, p_2] = 1$ and $B[p_1 + j', p_2 + j'] = 1$. Assume this were true. Then, some instance i is corrupted because a symbol is pushed on the stack during the pass l_1 over v_{p_1} and popped during the pass l_2 over v_{p_2}, and another instance i' is corrupted because a symbol is pushed on the stack during the pass l_1 over $v_{p_1 + j'}$ and popped during the pass l_2 over $v_{p_2 + j'}$. But in the pass l_1, v_{p_1} is visited before $v_{p_1 + j'}$, and in the pass l_2, v_{p_2} is visited before $v_{p_2 + j'}$. This contradicts the First-In Last-Out semantics of a stack.

Hence, on every diagonal $i_2 - i_1 = j$, the matrix B can contain a single 1. In total, B contains at most $\mathrm{O}\,(k)$ 1's. The cases where either pass l_1 or l_2 or both are right-to-left are treated similarly (except that we have to count the entries on the top-right to bottom-left diagonals when the directions of the two passes differ).

So, at most $\mathrm{O}(k \cdot r \cdot \log r)$ 4-tuples can corrupt instances, and each 4-tuple can corrupt at most $\mathrm{relsort}(\Phi_{k,m})$ instances. Therefore,

$$|\mathrm{BAD}(M', v')| \leq \mathrm{O}(k \cdot r \cdot \log r \cdot \mathrm{relsort}(\Phi_{k,m})).$$

4.3. Simulating a SM with a protocol. In this Section, we prove Lemma 4.2, which corresponds to claim (iii) in the outline from Section 2.

Let M' be a deterministic non-uniform SM with space bound s and pass bound r. Let $i^* \in [m]$ and let $\bar{y}_{-i^*}$ be a set of $m-1$ inputs to f. Our goal is to show that there exists a deterministic k-player NIH communication protocol $P' = P'(M', i^*, \bar{y}_{-i^*})$ that, on input x,

- if $i^* \in \mathrm{BAD}(M', v(i^*, x, \bar{y}_{-i^*}))$, then P' outputs "fail";
- otherwise, P' correctly simulates M' and outputs $M'(v(i^*, x, \bar{y}_{-i^*}))$;
- the cost of P' is $\mathrm{O}(k \cdot r \cdot \log(k \cdot r) \cdot s)$.

Fix an input x and let $v := v(i^*, x, \bar{y}_{-i^*})$. Observe that the players in P' share i^* and $\bar{y}_{-i^*}$, and their private inputs are $x_1, \ldots, x_k$. Thus, from the input v to M', they each know all instances $i \neq i^*$, and from instance i^*, player p only knows x_p. Furthermore, observe that the goal of P' is *not* to compute f, but rather, to compute the output of M' on v.

Private input symbols. We say that *an input symbol* from v *is private to player* p if it's part of $v_{p,\varphi_p^{-1}(i^*)}$, which is where x_p, the input to player p, is embedded in v. Input symbols that are not private to any player are *public*.

Private stack symbols. Let Γ be the sequence of configurations of M' on v. Consider a configuration $\gamma \in \Gamma$ and look at the contents of the stack in γ. For every symbol ξ appearing on the stack, we say that *the stack symbol* ξ *is private to player* p if the input head was scanning an input symbol private to player p in the configuration prior to the transition in which ξ was pushed on the stack. A stack symbol that is not private to any player is *public.*

Input-private and stack-private configurations. We say that *configuration γ is input-private to player p* if the input head in γ is scanning an input symbol that is private to player p. We say that *configuration γ is stack-private to player p* if it is not input-private to player p, the transition out of γ is a pop transition and the top stack symbol in γ is private to player p. A configuration that is neither input-private nor stack-private is *public.*

Intuitively, player p is responsible for simulating the transitions out of configurations that are input-private or stack-private to itself. Note that, there exists a configuration that is input-private and stack-private to two different players if and only if $i^* \in \mathrm{BAD}(M', v)$.

Hollow view. We say that *player p sees a hollow view of the stack* in a configuration γ if player p knows:

(i) the stack height;
(ii) for every symbol on the stack, whether it is public, or the player to which it is private (without knowing the symbol itself);
(iii) all stack symbols that are public or private to itself; and
(iv) for every $p' \neq p$, the top stack symbol in any contiguous zone of symbols private to player p'.

We say that *a player sees a hollow view of the configuration γ* if it knows the state of the SM, the location of the input head, and it also sees a hollow view of the stack in γ.

Some simple facts. In the protocol P', the players simulate Γ transition by transition, always using hollow views of the configurations along the way. We observe the following facts.

(1) If a player p sees a hollow view of a configuration γ, then that player can determine whether γ is public, input-private

to some player (since it knows the location of the input head), or stack-private. There is a mild subtlety involving the latter, specifically, the definition of γ being stack-private involves knowing that the transition out of γ is a pop transition. To this end, observe that, by virtue of part (iv) of the hollow view of the stack, if γ is not input-private, then player p knows the transition out of γ, even in the case when the top stack symbol is private to another player.

(2) If a player sees a hollow view of a *public* configuration γ, then that player can compute a hollow view of the configuration following γ (denoted by $\mathrm{next}(\gamma)$).

(3) As long as $i^* \notin \mathrm{BAD}(M', v)$, if player p sees a hollow view of a configuration γ which is input-private or stack-private to player p, then player p can compute a hollow view of $\mathrm{next}(\gamma)$.

(4) If player p sees a hollow view of a configuration γ that is input-private to itself, it can detect whether γ is stack-private to another player $p' \neq p$.

The protocol. We now describe the protocol P'. The sequence Γ is split into several contiguous sections, each of which is of exactly one of the following three types:

(a) *Public sections.* These sections start with *some* public configuration (not just any), they consist exclusively of public configurations, and they extend up to, and including, the first configuration which is no longer public.

Inductively, by facts (1) and (2) above, if all players have a hollow view of the configuration at the beginning of a public section, they can all simulate the entire public section starting at that configuration. At the end of the section, the players give control of the simulation to that player to which the configuration reached is input-private or stack-private. All this is done without any communication.

(b) *Input-private sections.* Each such section starts with, and contains, a maximal sequence of input-private configurations to some player p, and it ends with the single configuration immediately following that sequence. The latter might be public, or input-private to a player other than p, or stack-private to any player (including p).

Inductively, by facts (1), (3), and (4) above, if player p sees a hollow view of the configuration γ at the beginning of a sequence of input-private configurations to itself, it can either detect that instance i^* is corrupted, in which case player p aborts the simulation and the protocol P' outputs “fail”, or simulate the entire section. Let γ' be the configuration at the end of the section. At this point, player p communicates:

(v) the state in γ';
(vi) the input head position in γ';
(vii) the stack height L' in γ';
(viii) the top stack symbol in γ';
(ix) the lowest stack height L'' achieved in a configuration between γ and γ'; and
(x) the stack symbol at level L''.

(c) *Mixed stack-private and public sections.* Each such section begins with a configuration that is stack-private to some player p and contains a mix of configurations that are either stack-private to p or public.

Let γ be the configuration at the beginning of such a section. Let p be the player to which it is stack-private, and assume this player has a hollow view of γ. Let γ' be the first configuration following γ which is either input-private to any player (including p), or stack-private to a player other than p. Inductively, by facts (1), (2), and (3) above, player p can simulate the entire sequence of configurations between γ and γ'.

The key technical point of this entire proof is determining the right place where a player stops a privately simulated section of type (c). It turns out that this is not γ': doing so could result in public symbols being placed on the stack which would only be known to player p, unless they would be subsequently communicated, which would affect the cost of the protocol. Furthermore, it turns out it is also a bad idea

> for player p to end this section at the first public configuration encountered: the subtlety here is that the stack level might go up and down, and each time it goes down a few private stack symbols are being popped from the same contiguous section of private stack symbols. This scenario would hurt the cost of the protocol, for we could only bound the amount of communication in terms of the *length* of the contiguous zones of private stack symbols, which can be large.
>
> Instead, player p, looking at the entire sequence from γ to γ', computes *the minimum possible stack level* for this section, and finds the last configuration γ'' preceding, and possibly equal to, γ' where the stack level is minimal. At this point, player p communicates (v)–(viii) for configuration γ''. Observe that, by definition of γ'', (ix) and (x) are equal to (vii) and (viii), respectively.

Having defined the types of simulated sections, we show that the players have enough information to carry out the simulation. Let A be the number of different sections in the simulation. We claim that for every $1 \leq a \leq A$, *all* players eventually see a hollow view of the configuration at the beginning of section a. We prove this by induction on a.

For $a = 1$, observe that the initial configuration is either public or input-private to player 1. In either case, all players see a hollow view of the stack, because the stack is empty.

Inductively, assume all players have a hollow view of the configuration γ at the beginning of section $a \geq 1$.

If the section is of type (a), by fact (2), all players see a hollow view of the configuration at the end of this section.

If the section is of type (b), let p be the player to which γ is input-private. Clearly, by fact (3), p itself sees a hollow view of the configuration γ' at the end of the section. After p communicates (v)–(x), each player $p' \neq p$ updates its view of the stack as follows: it truncates the stack at height L''; fills it to the height L' with symbols private to player p; to compute part (iv) of the hollow view, player p' obtains the top stack symbol in γ' from (ix), and the stack symbol at level L'' from (x). Taking into account (v) and (vi), now player p' sees a hollow view of γ'.

If the section is of type (c), let p be the player to which γ is stack-private. Let γ'' be the configuration where this section ends. Clearly, by facts (2) and (3), player p sees a hollow view of γ''. After p communicates (v)–(viii) at the end of the section, each player $p' \neq p$ updates its view of the stack as follows: it truncates the stack at height L'; and it updates the top stack symbol using (vii). Taking into account (v) and (vi), now p' sees a hollow view of γ''.

This completes the description of the protocol P'. If $i^* \notin \mathrm{BAD}(M', v)$, the players get to the final configuration, obtaining the output of M' on input v. If $i^* \in \mathrm{BAD}(M', v)$, one of the players detects this during a section of type (b), the simulation is aborted, and P' and outputs "fail".

The cost of the protocol. We now compute the amount of communication in P'. The number of sections of type (b) is $k \cdot r$, one for every player, in every pass. For $1 \leq a \leq k \cdot r$, let S_a be the a-th section of type (b) and let γ_a and γ'_a be the configurations at the beginning and at the end of S_a.

Fix $a < k \cdot r$, and look at the stack in γ'_a. It contains public stack symbols, and several contiguous zones of private stack symbols, at most one such zone for every previous section of type (b). Furthermore, and crucially, observe that

Claim 4.3. *The number of sections of type (c) in between S_a and S_{a+1}* equals *the number of different contiguous zones of private stack symbols from γ'_a (the end of S_a) from which a symbol is popped before γ_{a+1} (the beginning of S_{a+1}).*

Proof (of Claim 4.3). This critical property follows from the way sections of type (c) are terminated. Specifically, we claim it is impossible for two different sections of type (c) between S_a and S_{a+1} to pop symbols from the *same contiguous zone* of stack symbols private to some player p.

Assume, for the sake of contradiction, that there are two sections S'_1 and S'_2 between S_a and S_{a+1} that pop symbols from the same *contiguous* zone of stack symbols private to player p. In between S'_1 and S'_2, there can be no input-private configurations (by definition of S_a and S_{a+1}), and also no configurations stack-private to a player other than p (otherwise, the zones of private stack symbols corresponding to S'_1 and S'_2 would not be contiguous). Hence, all configurations between S'_1 and S'_2 are either public or stack-private to player p. Moreover, the stack level in S'_2 is strictly lower than in S'_1. Then, player p should not have ended section S'_1 so early, instead, it should have simulated the entire section of configurations between S'_1 and S'_2, and then it should have continued with S'_2 (i.e., the period from S'_1 through S'_2 should really be all one segment).

Clearly, for every contiguous zone of private stack symbols from which symbols are popped in between S_a and S_{a+1}, there must be *at least one* associated section of type (c). By the argument above, there will be *exactly one* such section of type (c). ☐

Let $\gamma_{k\cdot r+1}$ denotes the final configuration. Consider the $(k \cdot r) \times (k \cdot r)$ matrix B, where $B[a_1, a_2] := 1$ if, in between γ'_{a_2} (the end of S_{a_2}) and γ_{a_2+1} (the beginning of S_{a_2+1} or the final configuration), a private stack symbol is popped that was pushed in S_{a_1}; and $B[a_1, a_2] := 0$ otherwise.

CLAIM 4.4. *There are at most* $\mathrm{O}\left(k \cdot r \cdot \log(k \cdot r)\right)$ *1 entries in* B.

PROOF (of Claim 4.4). The matrix is 0 under the main diagonal, because for a symbol from S_{a_1} to be popped after S_{a_2}, we must have $a_1 \leq a_2$. Consider the diagonal $a_2 - a_1 = j$ for some $j \geq 1$. We claim that it is impossible to have $B[a, a+j] = 1$ and $B[a+j', a+j+j'] = 1$ for some a and some $0 < j' \leq j$. Assume $B[a, a+j] = 1$, so in between γ'_{a+j} and γ_{a+j+1} a symbol is popped that was pushed in S_a. But then, the stack in γ_{a+j+1} can no longer contain any symbols pushed in $S_{a'}$, for any a' such that $a < a' \leq a + j$. In particular, for $a' = a + j'$, all symbols pushed in $S_{a+j'}$ are no longer on the stack in γ_{a+j+1}, which precedes, or equals, $\gamma_{a+j+j'}$, Hence, it is impossible to have $B[a+j', a+j+j'] = 1$.

The argument above shows that the matrix B contains at most $k \cdot r/(j+1)$ 1 entries on the diagonal $a_2 - a_1 = j$. Hence, in total, it contains at most $\mathrm{O}\left(k \cdot r \cdot \log(k \cdot r)\right)$ 1 entries. ☐

Lastly, we observe the following.

Claim 4.5. *The number of sections of type (c) in between S_a and S_{a+1} is at most the number of 1 entries in column a of matrix B.*

Proof (of Claim 4.5). By Claim 4.3, every section of type (c) in between S_a and S_{a+1} is associated with a contiguous zone of stack symbols from γ'_a (the end of S_a) that are private to some player p. Since all private stack symbols pushed in $S_{a'}$ are contiguous on the stack (this is true at any point of the simulation), we see that there can be at most one section of type (c) for every section $S_{a'}$ such that symbols from $S_{a'}$ are popped in between S_a and S_{a+1}, or, equivalently, such that $B[a', a] = 1$. ☐

Putting everything together, first observe that sections of type (a) do not result in any communication. There are $k \cdot r$ sections of type (b), and by Claims 4.4 and 4.5, there are a total of $\mathrm{O}(k \cdot r \cdot \log kr)$ sections of type (c).

And the end of every section of type (b) or (c), items (v)–(x) are communicated. The full state takes $\mathrm{O}(s)$ bits, the input head position takes $\log N \leq \mathrm{O}(s)$ bits, the stack height takes $\mathrm{O}(s)$ bits by Fact 3.1. Thus, every communication consists of $\mathrm{O}(s)$ bits. The total cost of the protocol P' is thus $\mathrm{O}(k \cdot r \cdot \log kr \cdot s)$.

5. Consequences

Corollary 1.2 (Restated). *Let $\epsilon > 0$ and $\delta < 1/2$ be constants. There exists a constant $\alpha = \alpha(\epsilon) > 0$ such that the following holds. Let $m = m(n) := n^\alpha$, let $k = k(n) := 2$, and let $N := 2 \cdot m \cdot n$. Let $\Psi^*_{2,m} := \{\mathrm{id}, \psi^*_m\}$ for the permutation ψ^*_m defined in Fact 3.3. Let $F = F_N := \mathrm{LiftMix}\left(\mathrm{IP}_{2,n}, \mathrm{XOR}_m, \Psi^*_{2,m}\right)$.*

Every randomized SM that computes F with error δ requires space $s = \omega\left(N^{1/4-\epsilon}\right)$ or two-way passes $r = \omega\left(N^{1/4-\epsilon}\right)$.

Observe that, by Fact 3.3, the function $F \in \mathsf{LOGSPACE}$.

The proof of Corollary 1.2 mainly consists of setting the right parameters in order to apply Theorem 1.1. The conclusion follows from the known communication complexity lower bound $R_2(\mathrm{IP}_{2,n}) \geq \Omega(n)$ (Chor & Goldreich 1988), where recall that R_2 denotes the private-coin, randomized 2-party communication complexity, where the error is a constant smaller than 1/2.

Proof (of Corollary 1.2). If $\epsilon \geq 1/4$, there is nothing to prove. Otherwise, let $\beta := 1/4 - \epsilon > 0$. Since $0 < \beta < 1/4$, we get

$$0 < \frac{1}{1-2\beta} < \frac{1}{2\beta}.$$

Define α so that $\alpha + 1$ is in between the two quantities above. That is,

$$\alpha := \left(\frac{1}{1-2\beta} + \frac{1}{2\beta}\right) \cdot \frac{1}{2} - 1.$$

Then, $1/(1-2\beta) - 1 < \alpha$ and $\alpha < 1/(2\beta) - 1$. By algebra, these inequalities imply:

$$\beta(\alpha+1) < \alpha/2, \tag{5.1}$$

$$2\beta(\alpha+1) < 1. \tag{5.2}$$

Assume that there exists a randomized SM M that computes F with error δ with space bound $s = \mathrm{O}\left(N^{\beta}\right)$ and pass bound $r = \mathrm{O}\left(N^{\beta}\right)$. By Theorem 1.1, there exists a randomized communication protocol for $\mathrm{IP}_{2,n}$ with cost $\mathrm{O}\left(k \cdot r \cdot \log(k \cdot r) \cdot s\right)$ and error at most $\delta + \mathrm{O}\left(d\right)$, where $d = k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Psi^*_{2,m}\right)/m$.

Observe that

$$\begin{aligned}
\mathrm{O}\left(d\right) &= \mathrm{O}\left(\frac{k \cdot r \cdot \log r \cdot \mathrm{relsort}\left(\Psi^*_{2,m}\right)}{m}\right) & \\
&= \mathrm{O}\left(\frac{k \cdot r \cdot \log r}{\sqrt{m}}\right) & \text{(by Fact 3.3)}\\
&= \mathrm{O}\left(\frac{r \cdot \log r}{\sqrt{m}}\right) & \text{(since } k = 2\text{)}\\
&= \mathrm{O}\left(\frac{(2 \cdot m \cdot n)^{\beta} \cdot \log(2 \cdot m \cdot n)^{\beta}}{\sqrt{m}}\right) & \text{(since } N = 2 \cdot m \cdot n\\
& & \text{and } r = \mathrm{O}\left(N^{\beta}\right))\\
&= \mathrm{O}\left(\frac{n^{\beta(\alpha+1)} \cdot \log n}{n^{\alpha/2}}\right) & \text{(since } m = n^{\alpha}\text{)}\\
&\longrightarrow 0 & \text{(by Equation (5.1))}
\end{aligned}$$

Hence, eventually, $d < (1/2 - \delta)/2$, and the error in P becomes at most $\delta + (1/2 - \delta)/2 < 1/2$. By standard arguments, repeating P

a constant number of times and taking the majority vote, we get a protocol for $\mathrm{IP}_{2,n}$ with error at most 1/3.

Similarly, observe that

$$\frac{\mathrm{O}\left(k \cdot r \cdot \log kr \cdot s\right)}{n} = \frac{\mathrm{O}\left((m \cdot n)^{2\beta} \cdot \log n\right)}{n} = \frac{\mathrm{O}\left(n^{2\beta(\alpha+1)} \cdot \log n\right)}{n} \longrightarrow 0,$$

where in the limit we use Equation (5.2). Therefore, the cost of the protocol P for $\mathrm{IP}_{2,n}$ (even after being repeated a constant number of times) is $o\left(n\right)$. This contradicts the known communication complexity lower bound $R_2(\mathrm{IP}_{2,n}) \geq \Omega\left(n\right)$ (Chor & Goldreich 1988). □

Corollary 1.3 (Restated). *Let* $\ell > 4$, $\epsilon \geq 0$ *and* $\delta < 1/2$ *be constants. There exists a constant* $0 < \beta < 1$ *such that any randomized SM computing a* $(1+\epsilon)$ *multiplicative approximation of* Freq_ℓ *with error* δ *requires space* $s = \omega\left(N'^{\beta}\right)$ *or passes* $r = \omega\left(N'^{\beta}\right)$*, where* N' *denotes the input size.*

In order to prove Corollary 1.3, we use one more result, saying that an efficient SM A computing frequency moments can be transformed into an efficient SM B computing the permuted Set Intersection function $F_N = \mathrm{LiftMix}\left(\mathrm{pSetInt}_{k,n}, \mathrm{OR}_m, \Phi_{k,m}\right)$, for any permutation family $\Phi = (\Phi_{k,m})_m$. Subsequently, we apply Theorem 1.1 to obtain an efficient NIH protocol for the promise Set Intersection function for which we have the lower bound $R_k(\mathrm{pSetInt}_{k,n}) \geq \Omega\left(n/k\right)$ (Gronemeier 2009).

This following streaming reduction originates in Alon *et al.* (1999), where there are no permutations Φ to deal with. It was rewritten in Beame & Huynh-Ngoc (2008) to deal with the case of several external tapes, where the permutations Φ are needed. The version in here is a combination of the two: We need to deal with permutations, and we also have to perform it "online", because we have no external tapes in this model. In order to deal with the permutations Φ, we use non-uniformity.

Lemma 5.3. *Let $\ell > 1$, $\epsilon \geq 0$ and $0 < \delta < 1/2$ be constants. Let n, k, m, N, s, r, Φ be as in Theorem 1.1. Set $k = k(n) := \Theta\left((m \cdot n)^{1/\ell}\right)$. Let $F_N := \mathrm{LiftMix}\left(\mathrm{pSetInt}_{k,n}, \mathrm{OR}_m, \Phi_{k,m}\right)$.*

Assume there exists a randomized SM A with space bound s, pass bound r, and error δ that computes a $(1+\epsilon)$ multiplicative approximation of Freq_ℓ. Then, there exists a randomized non-uniform SM $B = (B_N)$ that computes F_N with space bound $s'(N) = s(N) + \mathrm{O}(\log N)$, pass bound $r'(N) = r(N) + 2$, and error δ.

Proof (of Lemma 5.3). Let v be an input of size N to B_N. First, we assume B_N has another special tape on which it can write an input to A and then simulate A. We describe what this input $\bar{a} = \bar{a}(v)$ is, and we show that B_N can compute $F(v)$ from the output of A on $\bar{a}$. Then, we explain how to deal with the fact that B_N has no extra tape, and the construction of $\bar{a}$ and the simulation of A on $\bar{a}$ have to be performed in parallel.

The SM B_N begins its computation by counting, in one pass, the number of 1's in v, and saving this number as the value C on $\log N$ bits of its work tape. In pass number 2, B_N moves the input head back to the left end of the tape.

We think of the input to each instance of $\mathrm{pSetInt}_{k,n}$ as consisting of n columns and k rows. The output is 1 if there exists an all-1 column and 0 otherwise. By the promise, we know the number of 1's in each column is either 0, 1 or k, and also that there is at most one all-1 column in each instance of $\mathrm{pSetInt}_{k,n}$ We think of v, the input to F_N, as consisting of $m \cdot n$ columns, n of them for each of the m instances.

If $C < k$, there cannot be an all-1 column in any of the m instances of $\mathrm{pSetInt}_{k,n}$, so B_N simply outputs 0. Also, if $C > m \cdot n$, there must be at least two 1's in some column of one of the m instances of $\mathrm{pSetInt}_{k,n}$ (there are only $m \cdot n$ columns in total). In this case, by the promise in pSetInt, there is an all-1 column, so B_N outputs 1. From now on, assume neither of the above easy cases occurs, so $k \leq C \leq m \cdot n$.

Next, B_N constructs an input $\bar{a}$ to A as follows. The bits in v are ordered first by player number $p \in [k]$, then by part $i \in [m]$ (recall, the m instances in v_p occur in the order $(\varphi_p(1), \ldots, \varphi_p(m))$), and finally by bit position $j \in [n]$.

> For every (p, i, j) such that $v_{p,i,j} = 1$,
> B_N appends an element with value $\mathrm{val}(i, j) := (\varphi_p^{-1}(i) - 1) \cdot n + j$ to $\bar{a}$.

The property of this construction is that *the values corresponding to the 1's from the same column of v are all equal and are different from the values corresponding to any other 1's.*

We assume B_N is given $\varphi_1^{-1}, \ldots, \varphi_k^{-1} \in S_m$ *as non-uniform advice*, that is, it is hard-coded in the state transition of B_N. To see this is possible without affecting the space bound of B_N, observe that the input for this computation is $(p, i) \in [k] \times [m]$ and the output is $\varphi_p^{-1}(i) \in [m]$. Thus, $\mathrm{O}(\log N)$ bits are enough to describe both an input and an output as part of a state of B_N. We have already assumed that the space bound is always at least $s(N) \geq \Omega(\log N)$.

Next, observe that $\bar{a}$ consists of exactly C numbers from $[m \cdot n]$, so it is encoded as a bit string of length $N' = \mathrm{O}(C \cdot \log mn) \leq \mathrm{O}(m \cdot n \cdot \log mn) = o(N)$, because $N = k \cdot m \cdot n$ and $k = \Theta\left((m \cdot n)^{1/\ell}\right)$. Hence, eventually, $N' \leq N$.

After constructing $\bar{a}$, B_N runs $A = A_{N'}$ on input $\bar{a}$. If $A_{N'}$ makes an error, which happens with probability δ, so does B_N. Otherwise, $A_{N'}$ outputs some value

$$V \in \left[\frac{1}{1+\epsilon} \cdot \mathrm{Freq}_\ell(\bar{a}), (1+\epsilon) \cdot \mathrm{Freq}_\ell(\bar{a})\right].$$

At this point, B outputs 0 if and only if $V \leq C \cdot (1+\epsilon)$ and B outputs 1 otherwise. We need to argue B is correct.

If $F(v) = 0$, then all m instances of $\mathrm{pSetInt}_{k,n}$ inside v are 0-instances, so, by the promise, each column has at most one 1. Hence, all numbers in $\bar{a}$ are distinct, and $\mathrm{Freq}_\ell(\bar{a}) = C$. On the other hand, if $F(v) = 1$, then there are $b \geq 1$ 1-instances in v and $m - b$ 0-instances. Again by the promise, of all the $m \cdot n$ columns in v, b of them have k 1's, and all others have at most one 1. Then, $\mathrm{Freq}_\ell(\bar{a}) = C - b \cdot k + b \cdot k^\ell \geq C + k^\ell - k$. Furthermore, if $C \cdot (1+\epsilon) < (C + k^\ell - k)/(1+\epsilon)$, the ranges of possible output values V of $A_{N'}$ are disjoint between the cases $F(v) = 0$ and $F(v) = 1$. Thus, B is correct to output 0 if and only if $V \leq C \cdot (1+\epsilon)$.

We see that the ranges are disjoint if and only if $k^\ell - k > \epsilon \cdot C \cdot (2+\epsilon)$. Since $\ell > 1$, for large enough k, $k^\ell - k > k^\ell/2$. We also know that $m \cdot n \geq C$. Thus, it is enough to have $k^\ell > 2 \cdot \epsilon \cdot m \cdot n \cdot (2+\epsilon)$, which we achieve by setting $k := \Theta\left((m \cdot n)^{1/\ell}\right)$. This completes the argument that B_N is correct.

Lastly, we turn to the issue that B_N does not have an extra tape to first write $\bar{a}$ and then simulate $A_{N'}$ on $\bar{a}$, but has to do these in parallel. To this end, B_N operates as follows. At every step, it keeps the three values $(p, i, j) \in [k] \times [m] \times [n]$, corresponding to the location of the input head in v, using $\log N$ bits on its work tapes. B_N also has a buffer of $\log(m \cdot n) \leq \log N$ bits, which it fills with the value $\mathrm{val}(i, j) \in [m \cdot n]$. B_N simulates $A_{N'}$ by using the buffer as the input tape of $A_{N'}$, using space $s(N')$ on its work tapes, and using its own stack as the stack of $A_{N'}$. At any point where $A_{N'}$ attempts to move the head left (respectively, right) and the corresponding value from $\bar{a}$ is not in the buffer, B_N pauses the simulation of $A_{N'}$, scans left (respectively, right) on its input tape until it finds the next 1 bit, and refills the buffer with the new value $\mathrm{val}(i, j)$. Observe that B_N uses at most as many passes as $A_{N'}$ during this simulation. In total, B_N makes at most $r(N') + 2 \leq r(N) + 2$ passes, and uses space $s(N') + \mathrm{O}(\log N) \leq s(N) + \mathrm{O}(\log N)$. ☐

PROOF (of Corollary 1.3). We set the parameters first. Let $\beta := (\ell - 4)/(5 \cdot (\ell + 1))$. Observe that $0 < \beta < (\ell - 4)/(4 \cdot (\ell + 1)) < 1$. By algebra, this implies

$$0 < \frac{1}{1 - (2/\ell) - 2\beta(1 + 1/\ell)} < \frac{1}{(2/\ell) + 2\beta(1 + 1/\ell)}.$$

Set α so that $\alpha + 1$ is halfway in between the two quantities above, that is,

$$\alpha := \left(\frac{1}{1 - 2/\ell - 2\beta(1 + 1/\ell)} + \frac{1}{2/\ell + 2\beta(1 + 1/\ell)}\right) \cdot \frac{1}{2} - 1.$$

Thus,

$$\frac{1}{1 - 2/\ell - 2\beta(1 + 1/\ell)} < \alpha + 1 < \frac{1}{2/\ell + 2\beta(1 + 1/\ell)}.$$

By algebra, the two inequalities above imply:

$$(1+\alpha)\frac{1}{\ell} + (1+\alpha)\beta\left(1+\frac{1}{\ell}\right) < \frac{\alpha}{2}, \tag{5.4}$$

$$2(1+\alpha)\frac{1}{\ell} + 2(1+\alpha)\beta\left(1+\frac{1}{\ell}\right) < 1. \tag{5.5}$$

Assume there exists a SM A approximating Freq_ℓ to within $(1+\epsilon)$ that has space bound $\mathrm{O}\left(N'^\beta\right)$, pass bound $\mathrm{O}\left(N'^\beta\right)$, and error δ. We derive a contradiction.

Set $m := n^\alpha$ and $k = k(n) := \Theta\left((m\cdot n)^{1/\ell}\right) = \Theta\left(n^{(1+\alpha)/\ell}\right)$. Set $\Phi_{k,m} := \Phi^*_{k,m}$, for the family of permutations $\Phi^*_{k,m}$ defined in Fact 3.4. By Lemma 5.3, there exists a randomized non-uniform SM $B = B_N$ for $F_N = \mathrm{LiftMix}\left(\mathrm{pSetInt}_{k,n}, \mathrm{OR}_m, \Phi^*_{k,m}\right)$ with space bound $s := s(N) = \mathrm{O}\left(N^\beta\right) = O\big((kmn)^\beta\big)$, pass bound $r := r(N) = \mathrm{O}\left(N^\beta\right) = O\big((kmn)^\beta\big)$, and error δ. By Theorem 1.1, there exists a randomized NIH communication protocol P for $\mathrm{pSetInt}_{k,n}$ with cost $\mathrm{O}\left(k\cdot r\cdot \log(k\cdot r)\cdot s\right)$ and error $\delta + \mathrm{O}\left(d\right)$ where $d = k\cdot r\cdot \log r\cdot \mathrm{relsort}\left(\Phi^*_{k,m}\right)/m$.

Observe that

$$\begin{aligned}
\mathrm{O}\left(d\right) &= \mathrm{O}\left(\frac{k\cdot r\cdot \log r\cdot \mathrm{relsort}\left(\Phi^*_{k,m}\right)}{m}\right) \\
&= \mathrm{O}\left(\frac{k\cdot r\cdot \log r}{\sqrt{m}}\right) && \text{(by Fact 3.4)} \\
&= \mathrm{O}\left(\frac{n^{(1+\alpha)/\ell}\cdot n^{(1+\alpha)(1+1/\ell)\beta}\cdot \log n}{n^{\alpha/2}}\right) && \text{(using } r = \mathrm{O}\left((k\cdot m\cdot n)^\beta\right) \\
& && \text{and replacing)} \\
&\longrightarrow 0 && \text{(by Equation (5.4))}
\end{aligned}$$

Thus, eventually $d < (1/2-\delta)/2$, and the error in P becomes at most $\delta + (1/2-\delta)/2 < 1/2$. By standard arguments, repeating P a constant number of times and taking the majority vote we get a similar protocol with error at most $1/3$.

Furthermore, observe that

$$\frac{\mathrm{O}\left(k\cdot r\cdot \log(k\cdot r)\cdot s\right)}{n/k} = \frac{\mathrm{O}\left(n^{2(1+\alpha)/\ell}\cdot n^{2(1+\alpha)(1+1/\ell)\beta}\cdot \log n\right)}{n} \longrightarrow 0,$$

where we first used $r, s \leq (k \cdot m \cdot n)^{\beta}$, replaced for n, and finally used Equation (5.5). Therefore, the cost of P, even after constantly many repetitions, is $o\,(n/k)$. This contradicts the known lower bound $R_k(\mathrm{pSetInt}_{k,n}) \geq \Omega\,(n/k)$ (Gronemeier 2009). □

6. Perspective

Our trade-off lower bounds for Stack Machines become unconditional lower bounds for the number of passes when restricted to logarithmic space. At first sight, sublinear polynomial lower bounds might seem weak. To put this into perspective, we show that, conditional on some widely believed complexity assumptions, Stack Machines with even a single pass over the input decide languages outside NC and outside POLYLOGSPACE.

Let $\mathcal{C}$ be the class of languages decidable by logarithmic space Stack Machines with *a single pass over the input.* Allender (1989) proves the following lemma (we include a proof for completeness).

LEMMA 6.1. *$\mathcal{C}$ contains all unary languages (tally sets) in* P.

PROOF (of Lemma 6.1). Let U be a unary language in P. By Cook (1971), there exists a logarithmic space Stack Machine M deciding U. We build another logarithmic space Stack Machine M' deciding U, but now using a single pass over the input. M' simulates M as follows. On input $x = 0^n$, M' makes one pass over the input tape, computing n. This is stored on a work tape in binary, using space $\mathrm{O}\,(\log n)$. The work tape of M' also has an $\mathrm{O}\,(\log n)$ bit counter that holds the position of the input head of M, initialized at 1. (We assume that M receives its input $x = 0^n$ on a read-only tape of size $n+2$, with exactly one blank to the left and right of x.) Henceforth, M' simulates M without accessing its own input tape. Instead, M' increases and decreases the counter to simulate right and left transitions of M, and it decides based on the value of the counter if the symbol that would be seen by M is a 0 or a blank (it is a blank if and only if the counter equals 0 or $n+1$). M' has additional $\mathrm{O}\,(\log n)$ space, so it is itself a logarithmic space Stack Machine. □

By Allender (1989), if $\mathsf{PSPACE} \subsetneq \mathsf{EXP}$ then there exist unary languages in $\mathsf{P} \setminus \mathsf{NC}$. By Lemma 6.1, this immediately implies the following.

Corollary 6.2. *If* $\mathsf{PSPACE} \subsetneq \mathsf{EXP}$, *then* $\mathcal{C}$ *contains a language not in* NC.

We also obtain[2] a similar result involving $\mathsf{POLYLOGSPACE}$ instead of NC.

Lemma 6.3. *If* $\mathsf{E} \not\subset \mathsf{PSPACE}$, *then there exists a unary language in* $\mathsf{P} \setminus \mathsf{POLYLOGSPACE}$.

Proof (of Lemma 6.3). Assume there exists $L \in \mathsf{E} \setminus \mathsf{PSPACE}$. Let $\mathrm{int} : \{0,1\}^* \to \mathbb{N}$ denotes the function which maps a bit string s to the nonnegative integer whose binary representation is s. Let $U = \left\{ 0^{\mathrm{int}(1x)} \;\middle|\; x \in L \right\}$.

First, we show that $U \in \mathsf{P}$. Let M_L be a Turing Machine deciding L in time $2^{\mathrm{O}(n')}$ where n' is the input size. We construct a Turing Machine M_U deciding U as follows. On input $y = 0^n$, M_U first computes n in binary, obtaining a string x such that $n = \mathrm{int}(1x)$. This takes $\mathrm{O}(n)$ time. Observe that $|x| \le \mathrm{O}(\log n)$. Also, by definition of U, $y \in U$ if and only if $x \in L$. Now, M_U simulates M_L on x, and accepts if and only if the latter accepts. This takes time $2^{\mathrm{O}(|x|)} \le 2^{\mathrm{O}(\log n)} = n^{\mathrm{O}(1)}$.

Second, we show that $U \notin \mathsf{POLYLOGSPACE}$. To do this, we prove that $U \in \mathsf{POLYLOGSPACE}$ implies $L \in \mathsf{PSPACE}$, contradicting the definition of L. So, let M_U be a Turing Machine deciding U is space $(\log n')^{\mathrm{O}(1)}$ where n' is the input size. We build a Turing Machine M_L deciding L as follows. On input x, M_L first computes $n = \mathrm{int}(1x)$, and stores it in binary on a work tape, on $1 + |x|$ bits. Observe that $x \in L$ if and only if $0^n \in U$. Next, the goal of M_L is to simulate M_U on input 0^n. However, M_L must perform this simulation without writing 0^n explicitly on a tape, which would take space superpolynomial in the length of its input x. Instead, M_L uses a $\mathrm{O}(\log n) \le \mathrm{O}(|x|)$ bit counter to keep track of the position of the input head of M_U, initialized at 1. (We assume

---

[2] Book (1974) shows similar results using similar types of arguments.

that M_U receives its input 0^n on a read-only tape of size $n + 2$, with exactly one blank to the left and right of x.) M_L increases and decreases the counter to simulate right and left transitions of M_U, and it decides based on the value of the counter if the symbol that would be seen by M_U is a 0 or a blank (it is a blank if and only if the counter equals 0 or $n + 1$). We know that M_U uses $(\log n)^{O(1)} \leq |x|^{O(1)}$ space, so in total, M_L uses $|x|^{O(1)}$ space. $\square$

Lemmas 6.1 and 6.3 immediately imply the following.

COROLLARY 6.4. *If* $\mathsf{E} \not\subseteq \mathsf{PSPACE}$*, then* $\mathcal{C}$ *contains a language not in* $\mathsf{POLYLOGSPACE}$.

Finally, we address the comparison between Stack Machines and the read-write stream algorithms of Grohe & Schweikardt (2005). These are Turing Machines that have internal tapes and a constant number of read-write external tapes, space is bounded on the internal tapes, and passes are bounded over *all* external tapes. There is an "obvious" simulation of a Stack Machine by a read-write stream algorithm with 2 external tapes: use the second external tape to simulate the stack. However, if the Stack Machine makes a few passes over its input tape, it does *not* follow that the read-write stream algorithm makes a few passes over both of its external tapes. The crucial point is that there is no bound on the number of passes the Stack Machine makes on its stack, but such a bound appears in the read-write stream algorithm simulating the Stack Machine. As a result, this *particular* simulation cannot be used to obtain trade-off lower bounds for Stack Machines from similar lower bounds for r/w Stream Algorithms.

More generally, recall the following fact.

FACT 6.5. (Hernich & Schweikardt 2008, Lemma 4.8) *Turing Machines with any constant number of external tapes, with space bound* s *(on internal tapes) and pass bound* r *(on external tapes) can be simulated in* $\mathsf{DSPACE}\,(r^2 \cdot s)$.

Together, Corollary 6.4 and Fact 6.5 show that, assuming $\mathsf{E} \not\subseteq \mathsf{PSPACE}$, a logarithmic space Stack Machine with a single pass over the input *cannot* be simulated by a Turing Machine with any constant number of external tapes, space bound $(\log n)^{O(1)}$, and pass bound $(\log n)^{O(1)}$.

7. Discussion

When considering logarithmic space Stack Machines, Corollaries 1.2 and 1.3 give lower bounds on the *number of passes over the input tape* that are only sublinear, and they become uninteresting if directly translated into *time* lower bounds. However, in light of Corollaries 6.2 and 6.4, we believe it is interesting to interpret these results *without* attempting to translate them into time lower bounds.

Logarithmic space Stack Machines are closely connected to combinatorial circuits. As such, we ask whether such machines restricted to a sublinear polynomial number of passes characterize (even partially) a natural circuit family. We point out that this family has the following properties: (i) the circuits are of polynomial size; (ii) the family can compute Parity and Majority; and (iii) the family can compute some languages outside NC (again, assuming $\mathsf{PSPACE} \subsetneq \mathsf{EXP}$).

Acknowledgements

We would like to thank Siavosh Benabbas, Toniann Pitassi and Charles Rackoff for helpful discussions. We would also like to thank the anonymous reviewers for the detailed remarks and suggestions, and for bringing to our attention Brandenburg (1977) and Book (1974). M.D. is supported by NSF grant CCF-0832797, P.P. is supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61150110582, 61033001, 61061130540, 61073174, 61250110577.

References

E. W. Allender (1989). P-uniform circuit complexity. *J. Assoc. Comput. Mach.* **36**(4), 912–928. ISSN 0004-5411.

N. Alon, Y. Matias & M. Szegedy (1999). The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.* **58**(1), 137–147.

S. Arora & B. Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press.

L. Babai, N. Nisan & M. Szegedy (1992). Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.* **45**(2), 204–232. ISSN 0022-0000.

P. Beame & D.-T. Huynh-Ngoc (2008). On the Value of Multiple Read/Write Streams for Approximating Frequency Moments. In *FOCS*, 499–508. IEEE. ISBN 978-0-7695-3436-7.

P. Beame, T. S. Jayram & A. Rudra (2007). Lower bounds for randomized read/write stream algorithms. In *STOC*, 689–698. ACM.

R. V. Book (1974). Tally Languages and Complexity Classes. *Information and Control* **26**(2), 186–193.

A. Borodin, S. A. Cook, P. Dymond, L. Ruzzo & M. Tompa (1989). Two Applications of Inductive Counting for Complementation Problems. *SIAM J. Comput.* **18**.

F.-J. Brandenburg (1977). On one-way auxiliary pushdown automata. In *Theoretical Computer Science, 3rd GI-Conference, Darmstadt, Germany, March 28-30, 1977, Proceedings*, H. Tzschach, H. Waldschmidt & Hermann K.-G. Walter, editors, volume 48 of *Lecture Notes in Computer Science*, 132–144. Springer. ISBN 3-540-08138-0. http://dx.doi.org/10.1007/3-540-08138-0_11.

Jian-er Chen & Chee-Keng Yap (1991). Reversal complexity. *SIAM J. Comput.* **20**(4), 622–638. ISSN 0097-5397.

B. Chor & O. Goldreich (1988). Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM J. Comput.* **17**(2), 230–261.

S. A. Cook (1971). Characterizations of pushdown machines in terms of time-bounded computers. *J. Assoc. Comput. Mach.* **18**, 4–18. ISSN 0004-5411.

P. Erdös & G. Szekeres (1935). A combinatorial problem in geometry. *Compositio Math.* 2: 463-470.

M. Grohe, A. Hernich & N. Schweikardt (2006). Randomized computations on large data sets: Tight lower bounds. In *PODS*, 243–252. ACM.

M. Grohe & N. Schweikardt (2005). Lower bounds for sorting with few random accesses to external memory. In *PODS*, 238–249. ACM. ISBN 1-59593-062-0.

A. Gronemeier (2009). Asymptotically Optimal Lower Bounds on the NIH-Multi-Party Information Complexity of the AND-Function and Disjointness. In *STACS*, 505–516.

A. Hernich & N. Schweikardt (2008). Reversal complexity revisited. *Theor. Comput. Sci.* **401**(1-3), 191–205. ISSN 0304-3975.

Eyal Kushilevitz & Noam Nisan (1997). *Communication complexity.* Cambridge University Press, New York, NY, USA. ISBN 0-521-56067-5.

W. L. Ruzzo (1980). Tree-Size Bounded Alternation. *J. Comput. Syst. Sci.* **21**(2), 218–235.

W. L. Ruzzo (1981). On Uniform Circuit Complexity. *J. Comput. Syst. Sci.* **22**(3), 365–383.

M. Sipser (1997). *Introduction to the Theory of Computation.* PWS Publishing Company.

H. Venkateswaran (1991). Properties that characterize LOGCFL. *J. Comput. System Sci.* **43**(2), 380–404. ISSN 0022-0000.

Manuscript received 16 September 2010

Matei David
Center for Computational
Intractability,
Princeton University,
Princeton, NJ, USA.
`matei@cs.toronto.edu`

Periklis A. Papakonstantinou
ITCS, IIIS
Tsinghua University,
Beijing, People's Republic of China.
`papakons@tsinghua.edu.cn`