Communication Complexity for Asynchronous Systems of Finite Devices*

Tomasz Jurdziński
Department of Computer Science
Technical University of Chemnitz
and Institute of Computer Science
Wrocław University
tju@informatik.tu-chemnitz.de

Mirosław Kutyłowski
Mathematical Institute
Wrocław University of Technology
and Dept. of Mathematics and Computer Science
Poznań University
mirekk@im.pwr.wroc.pl

Jan Zatopiański
Institute of Computer Science
Wrocław University
jan.zatopianski@ii.uni.wroc.pl

Abstract

We consider systems consisting of a constant number of finite automata communicating via messages. We assume that the automata are asynchronous, but the answers given by the system must be always correct. We examine computational power of such systems by inspecting the number of messages exchanged. This is motivated by the fact that communication volume is one of the most important complexity measures.

We show that any asynchronous system of finite automata that exchanges $o(n)$ messages is able to recognize regular languages only. This is much different than in the case of synchronous systems considered before (where already a constant number of messages suffices to recognize some non-regular languages). We show that asynchronous and synchronous systems may differ significantly in their computational power also for tasks requiring $\Omega(n)$ messages.

We consider a language $L_{\text{trans}}$ consisting of words of the form $A \# A^T$, where $A^T$ denotes transposition of matrix $A$ and the matrices are written row by row. While it is easy to see that $L_{\text{trans}}$ can be recognized with $O(n)$ messages by a synchronous system of finite automata, we show that $L_{\text{trans}}$ requires $\Omega(n^{3/2}/\log^2 n)$ messages on any asynchronous system.

1 Introduction

One of the significant problems in emerging new areas of theoretical computer science is to understand capabilities of asynchronous computing. This is especially important in the case of systems that work in a distributed way and are severely limited in resources.

In this paper, we consider the simplest asynchronous system one can imagine. Nevertheless, it turns out that already in this setting we may see many interesting phenomena showing computational power hidden in synchronization.

Model We consider systems consisting of a constant number of finite automata working in parallel on a common read-only input data and cooperating by sending and receiving messages. In such a system, a common clock may synchronize their work so that all automata commence and terminate execution of a step at the same time. In this case, we talk about a synchronous system of finite automata (ASFA). If there is no such a clock, and a step of an automaton may take an arbitrary amount of time (independent of the duration of the previous steps and the speed of the other automata), then we talk about an asynchronous system of finite automata (ASFA).

We assume that each automaton of the system receives the messages sent by other automata instantly. Since the number of automata is constant, we may assume that the messages are broadcast, that is, delivered to all automata – this does not change the number of messages by more than of a constant factor. Each automaton has a number of buffers of a constant size, each buffer corresponding to messages coming from a single automaton. If the incom-

---

*This research was partially supported by Komitet Badań Naukowych, grant 8 T11C 032 15, Deutsche Forschungsgemeinschaft grant 2138-511301-54, and our industrial partner AxIT Polska.
ing messages fill the whole buffer and the next message is coming, we may assume that either the messages that do not fit into the buffer are lost or that the buffer works as a shift register with the oldest message removed. For each of these settings, we get the same bounds in this paper.

Each computation must terminate. At the end of a computation some automaton halts the system by sending a message that also determines whether the input has been accepted or not. Since for an asynchronous system, for the same input many communication patterns are possible, we say that the system accepts language \( L \) if and only if for \( x \in L \) every computation on \( x \) accepts, and for \( x \not\in L \), no computation accepts. That is, a computation should yield a correct answer, no matter what timing pattern occurs.

Each of the automata of the system is a deterministic two-way finite automaton with a single read head for the input tape. Each automaton starts the computation on the leftmost cell of the input tape. During a step, an automaton may write messages into buffers, through which it communicates with other automata, and read from the buffers storing messages from other automata. The transition function of an automaton depends on the current internal state, the symbol seen on the tape by the read head of the automaton, and the oldest message in each buffer. Within a step, the automaton enters a new internal state, moves the read head by at most one position on the input, removes the oldest message from each of the buffers through which it receives messages from other automata, and, depending of its new state, may send a message. More formally:

**Definition 1** A system \( S \) of finite automata \( A_1, \ldots, A_k \), is a tuple of the form \((Q, Q^{\text{start}}, F, \Sigma, \Delta, \delta)\). In this expression \( Q = (Q_1, \ldots, Q_k) \), where \( Q_i \), for \( i \leq k \), denotes a finite set of states of \( A_i \); \( Q^{\text{start}} = (q_1^{\text{start}}, \ldots, q_k^{\text{start}}) \) is a set of initial states of automata; \( F \) is a set of final states; \( \Sigma \) is a finite input alphabet; \( \Delta \) is finite alphabet of messages; \( \lambda \) is a special symbol meaning “no message” or “no move”; and \( \delta = (\delta_1, \ldots, \delta_k) \) is a tuple of transition functions of the automata. Each \( \delta_i \), transition function of \( A_i \), is a mapping \( \delta_i : Q_i \times (\Delta \cup \lambda)^{k-1} \times \Sigma \rightarrow Q_i \times (\Delta \cup \lambda)^{k-1} \times \{L, R, \perp\} \). Each automaton \( A_i \) has finite buffers \( B_{i,1}, \ldots, B_{i,1-1}, B_{i,1+1}, \ldots, B_{i,k} \) for messages coming from, respectively, \( A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_k \). A single step of automaton \( A_i \) is executed as follows. Assume that \( A_i \) reads a symbol \( u \) from the input tape and the (oldest) messages from the buffers \( B_{i,1}, \ldots, B_{i,1-1}, B_{i,1+1}, \ldots, B_{i,k} \) are \( \mu_1, \ldots, \mu_{i-1}, \mu_{i+1}, \ldots, \mu_k \). If there is no message waiting in a buffer \( B_{i,j} \), then we put \( \mu_j = \perp \). If \( q \) is the internal state of \( A_i \) and \( \delta_i(q, \mu_1, \ldots, \mu_k, u) = (q', \psi_1, \ldots, \psi_k, r) \), then: \( A_i \) changes its internal state to \( q' \), the messages read from the buffers are removed, for \( j \neq i \), if \( \psi_j \neq \perp \), then \( A_i \) sends \( \psi_j \) to the buffer \( B_{j,j} \), the read head of \( A_i \) moves on the input tape one position to the left (if \( r = L \)), or to the right (if \( r = R \)), or it stands idle (if \( r = \perp \)).

**Message complexity** In this paper, we consider the number of messages sent by a system as a complexity measure of the system. Since there are finitely many configurations on which a message may depend, we may assume that each message has a constant size. In this way, the number of messages corresponds roughly to the total volume of messages.

We denote by \( A - MES(f(n)) \) (\( MES(f(n)) \) respectively) the class of languages that may be recognized by an ASFA \( \text{(SSFA appropriately)} \) system such that at most \( f(n) \) messages are sent for any input of size \( n \). More precisely, if \( L \in A - MES(f(n)) \), then there exists an ASFA system \( M \) such that on an input of length \( n \) system \( M \) yields the answer whether \( x \in L \) using at most \( f(n) \) messages for any computation that may occur. It is interesting to see that for asynchronous systems already \( O(1) \) messages suffice to recognize some non-trivial languages. A simple example is the language \( L = \{a^n b^n | n \in \mathbb{N} \} \), which requires one message and two (synchronous) automata. It has been shown [7] that there is a dense hierarchy of languages for message complexity between \( \Theta(1) \) and \( \Theta(n) \). Even for languages requiring a constant number of messages, there is a hierarchy where each additional message makes the class of languages larger [7]. An interesting phenomenon is that there is no language with message complexity \( \omega(1) \) and \( o(\log \log \log n) \) [8]. It has been also shown that there is a tight hierarchy of functions which require \( \omega(n) \) messages for inputs of size \( n \). Such a case inspected in [7] concerns multiplication of matrices.

An asynchronous system may be synchronized with additional messages. Assume for example that we have a synchronous system with automata \( S_1, S_2 \) and we would like to emulate them by asynchronous automata \( A_1, A_2 \). After \( A_1 \) executes step \( i \) of \( S_1 \), it sends a message to \( A_2 \) and enters a state in which it waits for a response from \( A_2 \). Simultaneously, \( A_1 \) remembers the state of \( S_1 \). The message sent to \( A_2 \) either has empty contents or contains a message that \( S_1 \) sends at this moment. After receiving the message from \( A_1 \), automaton \( A_2 \) executes step \( i \) of \( S_2 \). Once it is done, it sends a message back to \( A_1 \) in a similar manner and enters a state in which it waits for a response of \( A_1 \).

The technique presented above, called later step-by-step synchronization, has one major weakness. \( \Omega(t) \) messages are necessary for simulating \( t \) steps of a synchronous system. So, even if a synchronous system is message efficient, the asynchronous system, constructed in this way, has to generate a lot of messages.

**Problem statement** The main problem that we address in this paper is whether it is possible to design message efficient asynchronous systems of finite automata that would perform better than those obtained through a step-by-step synchronization described above. The results that we obtain
witness weakness of asynchronous systems. They suggest that all we can do is to perform step by step synchronization with additional messages.

As far as we know, the problem of communication complexity in asynchronous systems has not been considered widely in the literature. On the other hand, it is known that timing in a parallel systems can be a hidden source of information. This has been used by many authors for tricky, unexpected results (see for instance [3]). However, such solutions cannot be used in a distributed environment, which is asynchronous by its nature. Also, there are relatively few results on systems (even synchronous ones) consisting of many processing units with limited resources. Perhaps, the most prominent direction here is the one based on the techniques of Borodin and Cook (see [2] and [10, 1]).

1.1 New results

The first result is that no asynchronous system with a sublinear number of messages may recognize a non-regular language, even though, many non-regular languages may be recognized by synchronous systems with a constant number of messages [7]. Our claim follows from two propositions presented below.

Proposition 2 $A - MES(O(1))$ contains only regular languages.

Proposition 3 [6] There is no language $L$ such that $L \in A - MES(o(1))$ and $L \in A - MES(o(n))$.

The above propositions can be proved with standard tools of complexity theory $^1$. In order to save space for technically more interesting results, we confine ourselves to a few comments on the proof techniques used. The main idea is to split the input into a constant number of “silent” blocks (i.e., blocks such that no message is sent during the computation when any automaton has its head inside such a block) and communication positions (i.e., positions scanned by some head when a message is sent). For the first proposition, one can apply essentially the same construction as for the proof that a two-way finite automaton may be simulated by a nondeterministic one-way automaton: the communication positions and behavior for each silent block are guessed, and simultaneously it is checked whether the guesses are correct. We show that for each input, there is an asynchronous computation that may be simulated in this way. For the second proposition, we consider any input with a sublinear number of messages and take the silent blocks of a nonconstant size. Then we find a way to compress the size of each of these blocks into a constant one so that for some asynchronous executions the system gives the same answer and the same number of messages is sent. For the “compressed input” the number of messages is $\Omega(n)$, since each silent block has a constant size.

Propositions 2 and 3 show that there is a substantial difference between capabilities of synchronous and asynchronous systems when a sublinear number of messages is concerned. How does it look like, for languages of at least linear complexity? Again, we show that asynchronous systems are much weaker and the number of messages might increase substantially. We demonstrate this phenomenon on the language $L_{trans}$ consisting of the words of the form $U^T \# V$, where $U^T$ denotes transposition of binary matrix $U$ and the matrices are written row by row:

$$L_{trans} = \{ w_1 \# w_2 : w_1, w_2 \in \{0, 1\}^* \}$$

where $w_i$ denotes the symbol in string $w$ on position $i$. Using a standard argument (see the full version of the paper) one can show that:

**Proposition 4** Language $L_{trans}$ has message complexity $\Theta(n)$ on synchronous systems.

By using the step-by-step synchronization technique to the algorithm from Proposition 4, we see that one can recognize $L_{trans}$ with $O(n^{3/2})$ messages on ASFA. The main technical result presented in this paper is a lower bound on message complexity of $L_{trans}$ on asynchronous systems that almost matches the above upper bound:

**Theorem 5** Language $L_{trans}$ requires $\Omega(n^{3/2} / \log^2 n)$ messages on ASFA.

The rest of the paper is devoted to the proof of Theorem 5, which is technically the most challenging and interesting part of our considerations. The reader should be not surprised that the lower bound proof is tedious. We must go beyond the standard information-theoretic and communication complexity argument, since in $O(n)$ messages one may encode the whole information about a half of the input word. So, our bound says that in this sense the asynchronous systems are very wasteful: each piece of information is sent many times, in most cases for synchronization purposes only.

Finally, let us mention that a related result [4] for matrix transposition has been obtained for one-tape Turing machines.

2 Fundamental properties

2.1 Simple computations

We may distinguish some situations in which we say that an automaton is in a waiting state. It occurs, if for an input

---

$^1$for a full proof see http://www.ii.uni.wroc.pl/~zato/papers/ccasfd-full.ps.gz
symbol seen and a given internal state, the automaton cannot make any move unless a message arrives in one of its buffers. All other states are called non-waiting states. We may assume that every automaton sends a message when it reaches a waiting state. Indeed, since there is no situation that all automata are in waiting states, this may increase the number of messages at most \( k \) times. (In order to avoid loosing some messages we may need to increase sizes of buffers \( k \) times).

Throughout the paper we consider so called simple computations. In such an execution mode, we assume that exactly one automaton makes a progress at a moment while the other automata stay idle. The active automaton is the automaton in a non-waiting state with the lowest index. This makes the computation fully deterministic and may be regarded as one of possible asynchronous executions. An active automaton during a simple computation behaves like a “standard” finite automaton, since no new message arrives until another automaton wakes up. This may happen only after the active automaton sends a message or reaches a waiting state (which again, by our assumption, means sending a message) and terminates to be active. Since the correct answer should be given for any asynchronous execution, this holds also for simple computations. Our general strategy is to fool automata while they execute a simple computation.

### 2.2 Transitions

A transition set of a word describes behavior of automata of the system during a simple computation on a word provided that no communication occurs:

**Definition 6** Let \( \{A_1, \ldots, A_k\} \) be a set of finite automata, and \( Q \) be the set of their internal states. A transition is a tuple \((A_i, q, b, s, q', s')\) where \( q, q' \in Q \) and \( s, s' \in \text{[Left, Right]} \) and \( b \) denotes contents of the buffers through which \( A_i \) gets messages from other automata. We say that a word \( w \) satisfies transition \((A_i, q, b, s, q', s')\), if automaton \( A_i \), starting the computation in the state \( q \) on the side \( s \) of \( w \) with the buffers containing \( b \), leaves the word \( w \) in the state \( q' \) on the side \( s' \) without sending any message (we assume that during this computation no message arrives from other automata). The **transition set** of \( w \) is a set of all transitions satisfied by \( w \).

Note that for asynchronous systems we do not concern the time spent by automata on \( w \). For synchronous systems this is not the case – duration of a computation on a word is an information that may play an important role for accepting or rejecting a word (compare [7]). For asynchronous systems, there is no a clock, so it does not make sense to measure duration of a transition through a block during a simple computation.

### 2.3 Random inputs

In this paper, we often use arguments based on Kolmogorov complexity [11]. For a string \( x \) over a binary alphabet, Kolmogorov complexity of \( x \), \( K(x) \), denotes the size of a smallest program (for a Turing machine) encoded as a binary string that outputs \( x \) on the empty input. For a string \( w \), Kolmogorov complexity of \( x \) conditioned on \( w \), \( K(x|w) \), denotes the size of a smallest program (for a Turing machine) encoded as a binary string that outputs \( x \), given input \( w \). It is a fundamental fact that for every \( c > 0 \), \( K(x) \geq |x| - c \log |x| \) where \( |x| \) is the length of \( x \), holds for an overwhelming majority of binary strings \( x \). Such strings are called random in the sense of Kolmogorov complexity, or simply random. Often, we say that such strings are incompressible, since any compressed version of \( x \) would lead to a small Turing machine outputting \( x \).

It is well known that Kolmogorov complexity is an extremely powerful tool in complexity theory (for some examples see [11]). In this paper, we use the following result on random inputs for finite automata. It says that if some state of the automaton is reachable, then it must occur pretty fast.

**Lemma 7** Let \( M \) be a deterministic finite automaton and \( x \) be a word of length \( n \) over alphabet \( \{0, 1\} \) such that \( K(x) > n - c \log n \) for some constant \( c \) and \( n \) large enough. Let \( E \) be a configuration of \( M \) on \( x \) such that starting from \( E \) the automaton \( M \) does not get into an infinite loop inside \( x \). There exists a constant \( c' \) (which depends only on \( c \)) such that for every state \( q \) of the automaton \( M \) which may be reached by \( M \) after leaving \( x \) (when \( M \) starts from \( E \) and \( x \) is a part of a larger input) the following property holds: If automaton \( M \) starts a computation in configuration \( E \) on input \( x \), then \( M \) reaches state \( q \) for the first time in at most \( c' \log n \) steps or it does not reach state \( q \) until it moves out from \( x \) and reads a symbol which is different from 0 and 1.

**Proof** The proof is based on the technique from [5] and will be included in the full version of the paper.

### 3 Complexity of \( L_{trans} \) on asynchronous systems

Before we go into details, let us present an outline of the proof.

Let \( C \) be a \( 3N \times 3N \) matrix with elements from the alphabet \( \{0, 1\} \). By \( C_i \) we denote the \( i \)th row of \( C \) and by \( C' \) we denote the \( i \)th column of \( C \) for \( i = 1, \ldots, 3N \). Let \( \text{center}(C) \) denote the submatrix of \( C \) which consists of the common part of the rows \( C_{N+1}, \ldots, C_{2N} \) and the columns \( C_{N+1}, \ldots, C_{2N} \).

Assume that an asynchronous system \( S \), which consists of \( k \) automata, recognizes the language \( L_{trans} \).
The proof is based on the analysis of simple computations of the system $S$ on random data. For sufficiently large $N$, assume that $U$ is a random $3N \times 3N$ matrix, (i.e., $K(U) > 9N^2 - c \log N$ for some fixed constant $c$; by $K(U)$ we mean $K(U_1, ..., U_N)$), $V = U^T$ is a transposition of $U$. Let $U = \text{center}(U)$ and $V = \text{center}(V)$. So, we consider a simple computation of system $S$ on the input word $w = U_1 \ldots U_{3N}$. Obviously, this input word must be accepted by $S$.

In our proof, we will use communication complexity results. So, first we construct communication protocols $P_1, P_2, ..., P_N$ for the problem of equality of binary vectors of length $N$. Namely, $P_i$ checks that $U_i = V^i$, where $U_i$, according to our notation is the $i$th row of $U = \text{center}(U)$ and $V^i$ is the $i$th column of $V = \text{center}(V)$. The protocols are based on the computations of the system $S$ on the input word $w$. As a protocol for the equality problem, each $P_i$ requires $\Omega(N)$ symbols of communication during work on a random $U_i, V^i$ (for a general bound for equality problem see [9]). Moreover, to every protocol $P_i$ we assign $\Omega(N/\log N)$ messages sent during the computation of the system $S$ on $w$. Let messages assigned to any of these protocols be called protocol-related messages. Others messages are called auxiliary messages. Our assignments fulfill two important properties:

1. each message of system $S$ may be assigned to at most $k$ protocols,

2. there are $\Omega(N/\log N)$ auxiliary messages between each $k^2$ consecutive protocol-related messages.

Combining these properties with the fact that we assign $\Omega(N/\log N)$ messages to each protocol and that there are $N$ protocols we get the lower bound from Theorem 5.

3.1 Protocol description

Let $i \in \{1, ..., N\}$. Below, we define communication protocol $P_i$. Assume that there are two parties, $X$ and $Y$, which have to verify whether an $N$-symbol vector $x$ (known only to $X$) is equal to an $N$-symbol vector $y$ (known only to $Y$). We provide a communication protocol for them.

We need here some additional notation. Let $UU$ be a submatrix of $U$ which consists of rows $U_{N+1}$ through $U_{2N}$. So, the $i$th row of $UU$ includes the $i$th row of $U$ for every $i \in \{1, ..., N\}$ (see Fig. 1).

Our communication protocol is based on simple computations of the system $S$. In the analysis of the protocol, we use the property stated in Lemma 7 and some notions based on this property. An obvious corollary of Lemma 7 is that starting from some configuration, the system $S$ (working on random matrices) sends a message in $O(\log N)$ steps or does not send a message until one automaton reaches a border, i.e., an end-marker or the # symbol in the middle. Using this fact, we may divide the periods of computation during which no message is sent into two categories.

Definition 8 (Long and short runs) We say that system $S$ makes a short run if starting from a communication configuration in at most $c' \log N$ steps a message is sent ($c'$ is the constant from Lemma 7). Otherwise, $S$ makes a long run.

Two notions of round and disclosure, presented below, are defined with respect to the number $i$ (so, their meanings are different for different protocols $P_i$). In both these definitions, we assume that the computation of the system $S$ on the input word $w^i$ is considered, where $w^i$ is an input word representing matrices $U$ and $V = U^T$, with $U_i$ and $V^i$ replaced by any words $x$ and $y$ of length $N$. So, both round and disclosure are parameterized by three parameters: $i$, $x$ and $y$. Let us warn the reader that the definition below is somewhat artificial, but nevertheless it serves quite well its purpose.

Definition 9 (Round) A round is a part of computation of $S$ which satisfies the following conditions:

1. at least one automaton has a head inside $U_i$ at the first step of the round,

2. no automaton scans $U_i$ at the step immediately before the beginning of the round,

3. there is at least one automaton with its head inside $UU_i$ during each step of the round,

4. there is no automaton with a head inside $UU_i$ immediately after termination of the round,

5. at least one message is sent during the round.
Note that the asymmetry between $U_i$ and $UU_i$ at the beginning and the end of a round is not accidental. Intuitive intention is to ensure that after finishing a round we need at least a linear (with respect to $N$) number of steps (and “almost a linear” number of messages) in order to begin a new round for the same protocol.

**Definition 10 (Disclosure)** Disclosure of $j$, $j \leq N$, is the following event that occurs during a round (so at least one automaton is inside $UU_i$):

1. the active automaton is in a short run and it scans the $i$th bit of row $V_j$ at this moment (so the $j$th bit of $y$)

2. no earlier configuration satisfies the above condition for the protocol $P_i$ and index $j$.

After disclosure of $j$ we say that $j$ is disclosed.

Intuitively speaking, disclosure of $j$ is the earliest moment at which the $j$th symbol of $x$ and the $j$th symbol of $y$ might be compared by the system. Indeed, then there is a head in the row of $V$ containing the $j$th element of $y$, and, simultaneously there is a head in the row containing $x$.

Now, we show how the protocol $P_i$ works. Namely, $X$ and $Y$ simulate the computation of $S$ on $w$ (i.e., with $U_i$ replaced by $x$ and $V^i$ replaced by $y$). We assume that both $X$ and $Y$ know a description of the system $S$ and the contents of $U \setminus U_i$, this is, all entries of $U$ without those of $U_i$. Observe that in this way they also know $V \setminus V^i$. The protocol looks as follows:

1. **Preprocessing:** $X$ sends to $Y$ the transition set of word $x$. Then $Y$ replies by sending the transition set of the word $v$ which is the shortest subword of the input which contains all bits of $V$ with $V^i$ replaced by $y$.

2. **After preprocessing,** $Y$ starts a simulation of $S$, without any communication with $X$. Then $Y$ proceeds until the beginning of the first round. Observe that $Y$ does not need to communicate with $X$ until this moment. When any automaton goes through $U_i$ without sending a message, $Y$ may use the knowledge on transition set of $x$ known from preprocessing. If a message is sent in a configuration in which any automaton is on $U_i$, a round starts (from transition set of $x$, $Y$ may deduce that a round begins).

3. **When a round begins,** $Y$ sends to $X$ a description of the current configuration (that is, the states of the automata, the contents of the buffers, and the positions of the heads).

4. **The computation within a round is simulated by $X.$** However, in some configurations during the round $X$ is unable to continue simulation, because $X$ lacks some information about the input which is known only to $Y$. Namely, it happens when one of the automata reads a symbol from $y$, say the $j$th symbol of $y$, and so far $Y$ has not informed $X$ about this symbol. Since $y$ is unknown to $X$, the further simulation is impossible without help of $Y$. First, $X$ sends a request containing the index, the position, and the internal state of the active automaton. Since we are working on random matrices, a short or a long run takes place. The answer that $X$ obtains from $Y$ is the following:

**Short run:** In this case, a disclosure of $j$ takes place. $Y$ answers by informing $X$ about disclosure and about the value of the $j$th bit of $y$.

**Long run:** In this case, $Y$ answers by saying at which border and in which state this long run terminates.

In both cases we assign the answer that $X$ obtains from $Y$ to the next message sent by the active automaton (so the message which yields a disclosure in a short run and the message sent by the active automaton after it reaches a border in a long run). A configuration from which a long run starts will be called advice-needed configuration.

5. **In the last configuration of the round,** $X$ sends to $Y$ information about the current configuration and the simulation is continued by $Y$ until the next round.

It is easy to check that using this protocol, $X$ and $Y$ are able to simulate the computation of the system $S$, and in this way check if $x = y$ (only in this case $S$ accepts). Let us remark here that in the case of a non-random $x$ our partition into long and short runs does not apply anymore. However, it is easy to modify the protocol in order to handle such cases.

### 3.2 Properties of the protocol

We assign the following messages of the system $S$ to the protocol $P_i$: the first message of each round, the messages sent at disclosures, and messages sent in advice-needed configurations. All these messages are called below protocol-related messages. The other messages issued by $S$ are called auxiliary.

Let us take a look at the protocol $P_i$ once again and inspect “when” messages between $X$ and $Y$ are exchanged. First, at the beginning and the end of each round the description of the current configuration is sent. Second, at disclosures, the position of the symbol of $y$ and its value is sent. Third, in advice-needed configurations, the descriptions of two configurations are exchanged. At the end of a round, again a full description of a configuration is sent. So, in each case, the messages exchanged have size $O(\log N)$. 

Corollary 11 The number of messages assigned to protocol $P_i$ is $\Omega(p/\log N)$, where $p$ is the number of bits exchanged between $X$ and $Y$.

For the next step we need the following lemma, which is a slight modification of communication complexity result for equality problem:

Lemma 12 Protocol $P_i$ needs $\Omega(N)$ bits of communication on inputs $x$ and $y$ such that $x = y = U_i = V^i$.

Proof From the previous assumption about $U$ and property of Kolmogorov complexity it follows that $K(U_i|\{U\setminus U_i\}) > N - c' \log N$, where $c'$ depends only on the constant $c$.

Assume that in protocol $P_i$ at most $N/4$ symbols are exchanged. We show how to compress $U_i$ (if $U\setminus U_i$ is known) below its Kolmogorov complexity. This yields a contradiction.

The new description of $U_i$ consists only of a description of the system $S$, a description of protocol $P_i$, and a sequence $\alpha$ of consecutive symbols exchanged between $X$ and $Y$ for inputs $x = y = U_i$ followed by a sequence $\beta$ of the same length such that if $\beta_i = 1$, then $\alpha_i$ is a symbol sent from $X$ to $Y$ and if $\beta_i = 0$, then $\alpha_i$ is a symbol sent from $Y$ to $X$. Descriptions of $S$ and $P_i$ have a constant length, $|\alpha_i| = |\beta_i| \leq N/4$. So, the length of the description of $U_i$ has length at most $N/2 + O(1)$ contradicting the assumption that $K(U_i|\{U\setminus U_i\}) > N - c' \log N$.

Finally, we check that we can recover $U_i$ from the above description. For this purpose, we simulate protocol $P_i$ on the side $X$ putting as $x$ all possible words of length $N$ (instead of $U_i$). For every word we check whether symbols send by $X$ agree with $\alpha$ and $\beta$. In the case that we need some information from $Y$, we get it from $\alpha$ (using information encoded in $\beta$). Observe that $\alpha$ is stored only for $x$, communication is encoded by $\alpha, \beta$ so that $X$ and $Y$ finish by agreeing that compared words are equal. Otherwise, the system $S$ would be faulty.\hfill \Box

By Lemma 12 and Corollary 11 we get:

Corollary 13 $\Omega(N/\log N)$ messages are assigned to each protocol.

Lemma 14 Each message sent during the computation of $S$ on $w$ may be assigned to at most $k$ protocols.

Proof Observe that if a message is assigned to protocol $P_i$, then at least one automaton has the head inside $UU_i$ at this moment. Since there are $k$ automata, the claim follows.\hfill \Box

Lemma 15 Let us consider $k^2$ consecutive protocol-related messages sent during the simple computation of $S$ on $w$. Then, between the first and the last message in this sequence, system $S$ sends $\Omega(N/\log N)$ auxiliary messages.

Proof First, recall that we may divide the computation of $S$ on random inputs into short runs and long runs such that each run starts after sending a message and finishes when a message is sent (see Definition 8 and Lemma 7). Recall that during a run only one automaton is active, since we consider simple computations only. Moreover, the active automaton makes $O(\log N)$ steps during the run or it moves to a border symbol.

Let $U_1, \ldots, U_n, V_1, \ldots, V_n$ be called sensitive blocks and $UU_1, \ldots, UU_n$ be called quasi-sensitive blocks (see Fig. 2). The following observation is crucial for the proof: after a run the distance of an automaton to a sensitive block is at least $N$ or changes during the run by $O(\log N)$ positions.

Let $M_1, \ldots, M_{k^2}$ be configurations of $S$ in which $k^2$ consecutive protocol-related messages are sent. We say that starting from a configuration $M_1$ automaton $A_i$ actively moves through a sensitive block $F$, if $A_i$ enters $F$ and a message is sent when $A_i$ is inside $F$. In particular, observe that during a long run $A_i$ does not move actively through any sensitive block. We say that a crucial event occurs for system $S$ between configurations $M_1$ and $M_{k^2}$ if at least one of the following conditions is satisfied:

1. some automaton actively moves through a sensitive block different from its nearest sensitive block in configuration $M_1$, or
2. some automaton leaves a quasi-sensitive block $R$ and afterwards it actively moves through the sensitive block contained in $R$.

The definition of a crucial event is somewhat technical, but observe that if a crucial event occurs, then some automaton $A_i$ actively moves through some sensitive block, starting from a configuration in which the distance between $A_i$ and this block is greater than $N$. So, $A_i$ needs $\Omega(N/\log N)$ runs. That means that $\Omega(N/\log N)$ auxiliary messages are needed, what gives the claim stated by Lemma 15. Therefore, we have to consider only the case in which the crucial event does not occur between configurations $M_1$ and
We show that in this situation either many auxiliary messages are sent or less than \( k^2 \) protocol-related messages occur. The last case cannot happen so this completes the proof of Lemma 15. Let us consider all kinds of protocol-related messages. Below we estimate the maximal number of such messages during the computation between configurations \( M_1 \) and \( M_{k^2} \).

**Round messages (the first messages in rounds):** There are at most \( k \) messages of this type. Indeed, each automaton may begin a new round at most once, since otherwise the first condition of a crucial event would be satisfied (if a round for a different \( U_i \) starts) or the second condition of a crucial event holds (if a round for the same \( U_i \) starts).

**Disclosure messages:** There are at most \((k-1)^2\) messages of this type. Indeed, each automaton in the right half of the input word \( w \) may cause a disclosure with an automaton staying on the left half at most once. Indeed, an automaton \( A_s \) with the head in the right half of the input cannot move through another sensitive block in the right half, since otherwise a crucial event would occur. So \( A_s \) may disclose only one value \( j \). Note that \( A_s \) cannot disclose \( j \) twice to another automaton \( A_u \). Indeed, in this case \( A_u \) would terminate one round and start a new one. This would cause a crucial event due to the first condition.

**Messages sent in advice-needed configurations:** There are at most \( k \) messages of this type or there are \( \Omega \left( N^3 / \log N \right) \) auxiliary messages. Indeed, if an automaton reaches a border symbol in a long run, then it requires \( \Omega \left( N^3 / \log N \right) \) runs in order to enter again a sensitive or quasi-sensitive block. On the other hand, an automaton has to be back in a sensitive or a quasi-sensitive block, if the next advice-needed configuration caused by this automaton occurs.

We see that either the claim of Lemma 15 is fulfilled, or the maximal number of protocol-related messages between \( M_1 \) and \( M_{k^2} \) is at most \((k-1)^2 + 2k < k^2\). In the last case we get a contradiction, so the lemma holds. \( \square \)

Finally, we conclude that Theorem 5 follows from the results of this section. Indeed, we only need to combine the results: by Corollary 13 and Lemma 14, there are \( \Omega \left( N^2 / k \log N \right) \) protocol-related messages during the computation of \( S \) on \( w \). By Lemma 15, during each part of computation in which \( k^2 \) consecutive protocol-related messages occur, we have also \( \Omega \left( N / \log N \right) \) auxiliary messages. That means that total number of messages is

\[
\Omega \left( \frac{N^2}{k \log N} \cdot \frac{1}{k^2 \log N} \right) = \Omega \left( \frac{N^3}{k^2 \log^2 N} \right).
\]

### 4 Conclusions

We see that asynchronous systems are quite weak and require a lot of communication. In fact, we know that in some cases have seen essentially all we can do is to “synchronize” the devices by sending messages back and forth.

Our proofs depend heavily on the assumption that whatever happens the system must provide a correct answer. However, this is not the only feasible way of using asynchronous systems. For instance, we may assume that the system has a randomized nature and the answer must be correct in most cases only. Many different ways in which asynchrony may occur deserve a separate examination. There is an ongoing research in this direction.

### References