Oblivious Collaboration

Yehuda Afek[1], Yakov Babichenko[2], Uriel Feige[3],
Eli Gafni[4], Nati Linial[5], and Benny Sudakov[6]

[1] The Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[2] Department of Mathematics, Hebrew University, Jerusalem, Israel
[3] Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot, Israel
[4] Computer Science Department, Univ. of California, Los Angeles, California
[5] School of Computer Science and Engineering, Hebrew University, Jerusalem, Israel
[6] Department of Mathematics, UCLA, Los Angeles, California

Abstract. We introduce *oblivious protocols*, a new framework for distributed computation with limited communication. Within this model we consider the *musical chairs* task $MC(n, m)$, involving n players (processors) and m chairs. Initially, players occupy arbitrary chairs. Two players are in conflict if they both occupy the same chair. The task terminates when there are no conflicts and each player occupies a different chair. Our oblivious protocols use only limited communication, and do so in an asynchronous fashion. Essentially, a player can only observe whether the player itself is in conflict or not, and nothing else. A player observing no conflict halts and never changes its chair, whereas a player observing a conflict changes its chair according to its deterministic program. Known results imply that even with more general communication primitives, no strategy of the players can guarantee termination if $m < 2n-1$. We show that even with this minimal communication termination can be guaranteed with only $m = 2n-1$ chairs. Our oblivious protocol can be extended to the well-known *Adaptive Renaming* problem, using a name-space that is as small as that of the optimal nonoblivious protocol.

We also make substantial progress in optimizing other parameters (such as program length) for our protocols, though many interesting questions remain open.

1 Introduction

In every distributed algorithm each processor must occasionally observe the activities of other processors. This can be realized by explicit communication primitives (such as by reading the messages that other processors send, or by inspecting some publicly accessible memory cell into which they write), or by sensing an effect on the environment due to the actions of other processors. Examples for the latter case are collision detection based algorithms for sharing Multi-Access broadcast media [19]. In our work, in analogy to the collision detection setting, we consider two severe limitations on the processors' behavior and ask how this affects the system's computational power: (i) A processor can

D. Peleg (Ed.): DISC 2011, LNCS 6950, pp. 489–504, 2011.

only post a proposal for its own output, (ii) Each processor is "blindfolded" and is only occasionally provided with the least possible amount of information, namely a single bit that indicates whether its current state is "good" or "bad". Here "bad/good" stands for whether or not this state conflicts with the global-state desired by the processor. Moreover, we also impose the requirement that algorithms are deterministic (use no randomization). This new minimalist model, properly defined, is called the *oblivious model*. This model might appear to be significantly weaker than other (deterministic) models studied in distributed computing. Yet, we show that two variants of the renaming problem, *adaptive renaming* (AR) (defined in [2]) and *musical chairs* (MC) (introduced in here) can be solved optimally within the oblivious model. Furthermore, we discuss the efficiency of oblivious solutions to these problems and the relations between the oblivious model and the wait-free asynchronous shared memory model with only reads and writes.

The current paper defines the oblivious model in general, but presents results only for the tasks MC and AR, and only with the collision predicate (which is natural for these tasks). We believe that the study of other tasks within the oblivious model can lead to additional interesting insights about the role of communication in distributed computing, though this is left to future work.

The *oblivious* model limits the operations available to individual processors. We find it convenient to model these limitations via a fictitious oracle. Associated with every state of a participating processor is a proposed output, though there could be several different states with the same proposed output. The state at which a processor halts thus defines its final output. The only way a processor can sense its environment is by querying the oracle about a single predicate on the current vector of outputs of the processors. Based on the single bit answer the processor needs to either halt with its current output, or proceed with its computation and propose a new output. But how can a processor's computation proceed? It has no information about the state of other processors (beyond the one bit that tells it that it must proceed), and we are forbidding randomization. Consequently, a processor's proposed output can depend only on its current state, and therefore the sequence of states that processor p_i traverses is simply an infinite word π_i over the alphabet of possible outputs. Upon receiving a negative answer from the oracle, processor p_i in state $\pi_i[k]$ moves to state $\pi_i[k+1]$. Given the definition of a computational task, it is up to the programmer to design the words π_i and the query that each processor poses to the oracle under which that task is always realized properly. Our only assumption is that the oracle correctly answers the queries, and a processor eventually halts/proceeds to the next state in his word upon a bad/good response from the oracle.

The *Musical Chairs*, $MC(n, m)$ task involves n *processors* $p_1, \ldots, p_n$ and, m *chairs* numbered $1, \ldots, m$. Each processor p_i starts in an arbitrary chair, dictated by the input. If the input chairs are all different, all processors are good and the input is the output. Otherwise, the task calls for each processor to capture a chair that differs from the chair captured by any other processor.

The *AdaptiveRenaming*(n, m) (AR(n, m)) task is a close relative of $MC(n, m)$. There are m slots (chairs) numbered $1, \ldots, m$ and each participant has to capture a different slot. The processors have no input. If only $k < n$ processors participate, then each has to capture (output) a different slot from the first $\min(2k - 1, m)$ slots. If all the n processors participate then each captures a different slot from the m slots.

In Section 2 we define the oblivious model in detail. For the MC and the AR problems we use the *collision* query – a processor is good iff it is the only one to propose the current chair. We show that in this case the general oblivious model simplifies considerably. These simplifications later help us produce an optimal solution. The infinite words (programs) considered here are an infinite repetition of a finite word.

Remarkably, for each processor we produce a program which is a single cyclic word (an infinite repetition of that word) on an alphabet of chairs. Furthermore, for the MC task the program can be started at any location in the word. This provides for self stabilization [11,12]. Namely, consider a system configuration where each processor occupies a different chair and there are no conflicts. Suppose that the system gets perturbed, and program counters change arbitrarily. This may create conflicts, but the system will nevertheless resettle obliviously in finite time into a conflict-free safe configuration.

Here are the main results presented in the current paper:

1. The introduction of the general oblivious model and its specialization to the problems at hand.
2. A proof that there are tasks that are solvable in a wait-free asynchronous shared memory model with only reads and writes, but not solvable obliviously.
3. The characterization of the minimal m for which there is an oblivious $MC(n, m)$ algorithm:

   Theorem 1. *There is an oblivious $MC(n, m)$ algorithm if and only if $m \geq 2n - 1$.*

   Moreover, for all $N > n$ there exist N words on m chairs such that any n out of the N words constitute an oblivious $MC(n, 2n - 1)$ algorithm.
4. The characterization of the minimal m for which there is an oblivious AR(n, m) algorithm:

   Theorem 2. *There is an oblivious $AR(n, m)$ algorithm if and only if $m \geq 2n - 1$.*

5. The words in Theorem 1 use the least number of chairs, namely $m = 2n - 1$. However, the length of these words is doubly exponential in n. Are there oblivious MC algorithms with much shorter words? Even length $O(n)$? Perhaps even length m? How long can the scheduler survive? Here we consider systems with $N \geq n$ words (programs) and any n out of the N should constitute a solution of MC. We call these $MC(n, m)$ systems with N words.

Theorem 3. *For every $N \geq n$, almost every choice of N random words of length $cn \log N$ in an alphabet of $m = 7n$ letters is an $MC(n, m)$ system with N* full *words (words that contain every letter in $1, \ldots, m$). Moreover, every schedule on these words terminates in $O(n \log N)$ steps. Here c is an absolute constant.*

6. Since we are dealing with full words (words that contain every letter in $1, \ldots, m$) and we seek to make them short, we are ultimately led to consider the case where each word is a permutation on $[m]$. At the moment the main reason to study this question is its aesthetic appeal. We can design permutation-based oblivious $MC(n, 2n-1)$ algorithms for very small n (provably for $n = 3$, computer assisted proof for $n = 4$). We suspect that no such constructions are possible for large values of n, but we are unable at present to show this. We do know, though that

Theorem 4. *For every integer $d \geq 1$ there is a collection of $N = n^d$ permutations on $m = cn$ symbols such that every n of these permutations constitute an oblivious $MC(n, m)$ algorithm. The constant c depends only on d. In fact, this holds for almost every choice of N random permutations on $[m]$.*

We should stress that our proofs of Theorems 3 and 4 are purely existential. The explicit construction of such systems of words remains largely open, though we do have some results in this direction, e.g.,

Theorem 5. *For every integer $d \geq 1$ there is an explicitly constructed collection of $N = n^d$ permutations on $m = O(d^2 n^2)$ symbols such that every n of these permutations constitute an oblivious $MC(n, m)$ algorithm.*

1.1 Related Work

Two variations of the renaming problem were introduced in [2], *Weak Renaming* and *Adaptive Renaming* (AR). In the former, there is an unbounded universe of processor ids of which n wake up and have to each select a different name in the range $\{1, \ldots M(n)\}$. In the AR$(n, M(n))$ problem, which is one of the two problems solved obliviously in this paper, the universe consists of n processors, $\{p_1, \ldots, p_n\}$ and again they have to each capture a different integer in the range $\{1, \ldots M(n)\}$. Yet in AR, if the size of the participating set is $k < n$, outputs are restricted to be in $\{1, \ldots, 2k-1\}$ (hence the algorithm is adaptive to the number of processors participating). The renaming algorithm presented in [2] solves both variants with $M(n) = 2n - 1$.

Weak renaming is solvable with $M(n) = 2n-2$ for infinitely many different n's, called "exceptional" [10]. For AR$(n, M(n))$, $M(n) = 2n - 1$ is a lower bound as shown in [15]. The proof of this lower bound builds upon previous impossibility results for *set consensus* (see [9,18,20]), by showing that given a hypothetical algorithm for AR$(n, 2n-2)$ in addition to read-write registers one can solve set consensus in a wait-free manner.

Both weak and adaptive renaming algorithms have been extensively studied over the last two and a half decades, but aside from the above mentioned works concerning solvability, all the studies are about complexity, which is not the subject of this paper.

The musical chairs problem is weakly related to the Musical Benches (MB) [16] problem. In MB there are n benches and $n + 1$ players. Each bench has two seats. Every player needs to occupy a seat, and more than one player can occupy the same seat. An output is legal if in every bench at most one seat is occupied. Initially, players occupy arbitrary seats. If the initial configuration is legal, it has to be the output. However, if the initial configuration is not legal, then players can move and must return a legal output. In [16] it is shown, using the Bursuk-Ulam Theorem, that the task for $n = 2$ has no wait-free solution in an asynchronous shared memory model with only reads and writes. MC shares with MB the flavor of the game of "jumping" from seat to seat. However, MC is about separating players from each other, whereas MB is about getting players on a bench to arrive at consensus.

There are other contexts in which algorithms of an oblivious nature were considered. An algorithm in which each process is assigned a permutation which specifies the order it is to do work is presented in [3]. Algorithms to compute a maximal independent set with only carrier sensing with or without collision detection are provided in this volume [6]. Among all work on algorithms with an oblivious nature, we find it most instructive to compare our work with work on *universal traversal sequences* (UTS) for covering graphs. A word over the alphabet $\{0, 1, \ldots d - 1\}$ can guide a walk on an n-vertex d-regular undirected graph: in each step the walk selects its next out-going edge according to the respective symbol of the word. Such a word is a UTS if for *every* connected n-vertex d-regular graph, regardless of how each vertex labels its out-going edge, the corresponding walk visits all vertices of the graph. In [1] it is shown that a sufficiently long random word (say, of length n^5) is almost surely a UTS. In analogy, our proof of Theorem 3 shows that sufficiently long random words almost surely form MC algorithms. However, in our case the proof needs to overcome an obstacle not present in the UTS case. The difference is that in MC, as words get longer, the scheduler also gets more choices of how to schedule them, whereas for UTS the number of graphs is fixed independently of the length of the words. As a consequence, for some range of parameters (e.g., provably when $m < 2n - 1$, as Theorem 1 shows), the statement is simply not true. There are no analogous forbidden ranges of parameters for universal traversal sequences.

1.2 Discussion

Due to space limitations, large parts (including most proofs) are omitted from the current version of this paper. The reader interested in more details is referred to [5].

A number of simple observations follow from the requirement that oblivious algorithms are deterministic. (i) An oblivious $MC(n, m)$ algorithm cannot include any two identical words. Otherwise the corresponding players might move

together in lock-step, constantly being in collision. Hence it is essential that no two processors have the same program. (ii) For every oblivious $MC(n, m)$ algorithm with cyclic words, there is a finite upper bound on the number of moves a processor can make before termination. This is because there are only finitely many system configurations (a system configuration is determined by one state for each processor, and the number of possible states of a processor is equal to the length of the cycle in its cyclic word, and hence finite), and in a terminating sequence of moves no system configuration can be visited twice. (iii) In fact, for every collection of finite words there is a directed graph whose vertices are all the system configurations. Edges correspond to the possible transitions. The collection of words constitute an oblivious MC protocol iff this graph is acyclic.

Not all aspects of oblivious protocols are required for the purpose of the lower bound $m \geq 2n - 1$. The two crucial aspects are the asynchrony of the model, and the fact that our algorithms are deterministic (no randomization). In a synchronous setting (where in every time step, every processor involved in a collision moves to its next state), $m = n$ suffices even for oblivious protocols. (This can be proven using the techniques of Theorem 3. Details are in [5].) Likewise, $m = n$ suffice if randomization is allowed – with probability 1 eventually there are no collisions. However, no specific upper bound on the number of steps can be guaranteed in this case. Moreover, if the randomized algorithms are run using pseudorandom generators (rather than true randomness) the argument breaks. For any fixed seed of a pseudorandom generator, the algorithm becomes deterministic and the lower bound $m \geq 2n - 1$ holds.

The lower bound of $m \geq 2n - 1$ uses some benign-looking aspects of the MC task, so further discussion is called for. Recall that each processor starts in an arbitrary chair, dictated by the input. In the absence of an external input specifying the starting chair, a trivial oblivious MC algorithm (with $m = n$) contains n distinct single-letter words. Another requirement is that if the input chairs are all different, all processors are good and the input is the output. Without such a requirement, the processors might simply ignore the initial input and the trivial oblivious MC algorithm would still apply. Hence the lower bound of $m \geq 2n - 1$ depends on requirements beyond the need for each processor to capture a different chair. Here this extra requirement is the possibility to dictate an output. This particular requirement makes it easy to transfer previously existing lower bounds to our MC problem.

Our present proof for the lower bound of $m \geq 2n - 1$ leaves something to be desired. It relies on previous nontrivial work in distributed computing. What's worse is that we prove a lower bound for a simple oblivious model via a reduction to a lower bound proved in a more complicated model. This roundabout approach obscures the essential properties that make the lower bound work. Indeed, in a companion manuscript (in preparation), we present a self contained proof for the lower bound of $m \geq 2n-1$. That presentation clarifies the minimal requirements that are needed in order to make the lower bound work. In particular, it is not necessary that one can dictate an arbitrary starting chair for each processor – dictating one of two chairs suffices.

As noted, we design oblivious $MC(n, m)$ protocols with $m = 2n - 1$. Part of our work also concerns analyzing what ratios between m and n one can obtain using collections of randomly chosen words as in Theorem 3. As explained in the introduction, this allows us to present more efficient deterministic oblivious programs – though random words seem to need more chairs, they can reach conflict free configurations more quickly. Moreover, the use of random words is a design principle that can be applied to design oblivious algorithms for other tasks as well. Developing an understanding of what they can achieve and techniques for their analysis is likely to pay off in the long run. One of the major questions that remain open in our work is whether randomly chosen words can be used to design deterministic oblivious MC protocols with $m = 2n - 1$.

2 The Oblivious Model

The *Oblivious* model (formally defined in [5]) is an asynchronous distributed computing model in which each processor, at each point of time, exposes an output value it currently proposes, and may receive at most one bit of information. This bit indicates whether its proposed output is legal with respect to the other currently proposed outputs (and hence the processor may halt) or not (and then the processor should continue the computation). If a processor decides to halt at the current state, then its proposed output is its final output. We denote the set of possible output values by O. A *system configuration* (or configuration for short) is a vector of n elements, one per processor, whose entries come from the set $O \cup \{\perp\}$. Here $\perp$ represents a processor that has not yet proposed any output, either because it is not participating, or because it was not scheduled yet to propose an output (these two cases are indistinguishable to other processors). An entry from O represents the output a corresponding processor proposes in the configuration. In an oblivious algorithm correctly designed for a given task, eventually all participating processors must halt, and the final configuration must be a legal output vector in the task specification.

The defining feature of the *oblivious* model is that each processor may receive only one bit of information about the system configuration in each computation step. Namely, for each processor there is a predicate that maps configurations to one of two values, one dictating that the processor will change its state, the other dictating that it should halt in its current state. In the most general setting the predicate provided for each processor may depend on its input. However, throughout an execution one predicate is used for each processor. A necessary (but not sufficient) condition for correct oblivious algorithms is that in every illegal configuration at least one processor's predicate dictates a change of state. Our formal model does not exclude the use of arbitrary complex predicates (as long as they depend only on the current configuration), but oblivious algorithms have greater appeal when the predicates involved are simple and natural. For the two tasks considered in this paper, the same *collision* predicate is used by all the processors.

Initially, and as a function of its input, each processor p_i selects a word π_i over O, and a predicate $pred_i$ on the set of of all configurations. The first letter

in π_i is p_i's input, i.e., $\pi_i[1] = input_i \in O$. For tasks such as AR in which a processor need not have any input, the first letter is set to be an output that is valid if no other processor participates (hence for AR the first letter is 1).

We describe the system using the notion of an omnipotent know-all scheduler called *asynchronous* (other schedulers with different names are described in the sequel). Execution under the control of the asynchronous scheduler proceeds in rounds. The scheduler maintains a set P of participating processors, a set $E \subset P$ of enabled processors, and a set $DONE$ (disjoint from P) of processors that have already halted. These sets are initially empty. In each round the scheduler performs the following sequence of operations. It may add some not yet participating processors to P. It may evaluate the predicate $pred_i$ for some subset of processors in $P \setminus E$. If $pred_i$ evaluates to true, the scheduler adds processor p_i to the set E. Otherwise, if it evaluates to false, it removes p_i from P and adds it to the set $DONE$. Finally, the scheduler selects a subset $SE \subseteq E$, removes it from E, and moves each $p_i \in SE$ to its next letter in π_i. I.e., the current output of p_i is replaced by the next one in its program, π_i. This completes the round.

An oblivious algorithm solves a task if for every input vector, the scheduler is forced to eventually place all participating processors in the $DONE$ set. At that point it can no longer continue, and the final configuration is such that $(v_{inp}, v_{out}) \in \Delta$, the relation that defines the task.

The asynchronous scheduler for oblivious algorithms mimics the behavior of a wait-free algorithm in an asynchronous shared memory model with only reads and writes, on configurations. Theorem 6 below is proved in [5] simply by having each processor emulate the scheduler through reads (snapshots) and writes of its newly proposed output in shared memory.

Theorem 6. *Every task that is solvable obliviously has a wait-free solution in an asynchronous shared memory model with only reads and writes.*

Thus the oblivious model is subsumed by the wait-free asynchronous shared memory model with only reads and writes. Is this a proper inclusion? To clarify the answer we introduce an intermediate class of tasks that we call Output Negotiation, or ON. It includes those tasks that have a wait-free solution in an asynchronous shared memory model with only reads and writes in a system where writing is in the oblivious model (processors can only expose their proposed outputs), whereas reading is as in the general wait-free asynchronous shared memory model with only reads and writes (a processor can read all exposed information rather than only a single predicate). By definition, every obliviously solvable task is ON solvable.

Corollary 7. *Every obliviously solvable task is in ON.*

Obviously, ON is a subset of the wait-free asynchronous shared memory model with only reads and writes, and in Theorem 8 below whose proof is in [5] we show that this inclusion is proper. In the proof we consider the task $AntiMC$ which is a variation on epsilon agreement [13] and show that AntiMC is not solvable just

by communicating outputs. AntiMC is a task with 3 processors whose input and output are each a number in $\{1, \ldots, 5\}$. A processor running alone must output its input. If more than one processor participates, all the outputs must lie within two consecutive numbers.

Theorem 8. *There exists a task, AntiMC, that has a wait-free solution in an asynchronous shared memory model with only reads and writes but does not belong to ON.*

2.1 Impossibility of $MC(n, 2n-2)$

In Sections 3 and 4 we show that $MC(n, 2n-1)$ and $\mathrm{AR}(n, 2n-1)$ are solvable obliviously. $\mathrm{AR}(n, 2n-2)$ has no wait-free solution in an asynchronous shared memory model with only reads and writes [14,15], and hence not solvable obliviously either. Theorem 9 whose proof is in [5] shows a reduction from $\mathrm{AR}(n, 2n-2)$ to $MC(n, 2n-2)$. This implies that $MC(n, 2n-2)$ has no wait-free solution in an asynchronous shared memory model with only reads and writes, and hence also not solvable obliviously.

Theorem 9. *$AR(n, 2n-2)$ is wait-free reducible to $MC(n, 2n-2)$ in an asynchronous shared memory model with only reads and writes.*

2.2 Cyclic Finite Programs or Words

For the AR task, processors have no input (or alternatively, are assumed to always have the input 1), and hence each processor has only one sequence. Our constructions of oblivious algorithms all have the property that the same sequence is used for all inputs. Moreover, we consider finite sequences over which the processor goes cyclically. In the MC task one can designate m locations in the word, each corresponding to a possible output that has been dictated by the input to the processor and each processor advances cyclically on the word starting from that designated location.

2.3 Simplified Oblivious Model for MC and AR

The use of the collision predicate can be shown to imply that for the AR and MC problems it is sufficient to consider a much simpler scheduler, the ***Pairwise Immediate scheduler***: In each round this scheduler selects two processors that are currently colliding with each other, and moves either one or both of them, c'est tout. Suppose that every processor has an associated word. We show that given an initial configuration (starting positions on the words), the oblivious asynchronous scheduler runs to infinity iff the pairwise immediate scheduler does. This scheduler is then used in constructions in Sections 3 and 4. The constructions in Section 5 use an even more restrictive scheduler, the ***Canonical scheduler***. Like the pairwise immediate scheduler, the canonical scheduler can move only one or both of two currently colliding processors, but unlike the pairwise immediate scheduler, the choice of which two colliding processors to consider is not made by the scheduler, but rather dictated to it. For formal definitions of these schedulers and the proof of their equivalence, see [5].

3 An Oblivious MC Algorithm with $2n-1$ Chairs

This section is dedicated to the upper bound that is stated in Theorem 1. The proof of this theorem is inductive and rather technical. For lack of space, the full text with all proofs is given in [5]. In what follows we attempt to give the reader a sense of the main ingredients of the construction and how they come together in the proof.

3.1 Preliminaries

The length of a word w is denoted by $|w|$. The concatenation of words is denoted by $\circ$. The r-th power of w is denoted by $w^r = w \circ w \ldots \circ w$ (r times). Given a word π and a letter c, we denote by $c \otimes \pi$ the word in which the letters are alternately c and a letter from π in consecutive order. For example if $\pi = 2343$ and $c = 1$ then $c \otimes \pi = 12131413$. A collection of words $\pi_1, \pi_2, ..., \pi_n$ is called *terminal* if no schedule can fully traverse even one of the π_i. Note that we can construct a terminal collection from any MC algorithm just by raising each word to a high enough power.

We now introduce some of our basic machinery in this area. A key tool is a method to extend terminal sets of words.

Proposition 1. *Let n, m, N be integers with $1 < n < m$. Let $\Pi = \{\pi_1, \ldots \pi_N\}$ be a collection of m-full words such that*

$$\textit{every } n \textit{ of these words form an oblivious } MC(n,m) \textit{ algorithm.} \qquad (1)$$

Then Π can be extended to a set of $N+1$ m-full words that satisfy condition (1).

proof. Suppose that for every choice of n words from Π and for every initial configuration no schedule lasts more than t steps. (By the pigeonhole principle $t \leq L^n$, where L is the length of the longest word in Π). For a word π, let π' be defined as follows: If $|\pi| \geq t$, then $\pi' = \pi$. Otherwise it consists of the first t letters in π^r where $r > |\pi|/t$. The new word that we introduce is $\pi_{N+1} = \pi'_1 \circ \pi'_2 \circ \ldots \circ \pi'_n$. It is a full word, since it contains the full word π_1 as a sub-word.

We need to show that every set Π' of $n-1$ words from Π together with π_{N+1} constitute an oblivious $MC(n,m)$ algorithm. Observe that in any infinite schedule involving these words, the word π_{N+1} must move infinitely often. Otherwise, if it remains on a letter c from some point on, replace the word π_{N+1} by an arbitrary word from $\Pi - \Pi'$ and stay put on the letter c in this word. This contradicts our assumption concerning Π. (Note that this word contains the letter c by our fullness assumption.) But π_{N+1} moves infinitely often, and it is a concatenation of n words whereas Π' contains only $n-1$ words. Therefore eventually π_{N+1} must reach the beginning of a word π_α for some $\pi_\alpha \notin \Pi'$. From this point onward, π_{N+1} cannot proceed for t additional steps, contrary to our assumption. □

Note that by repeated application of Proposition 1, we can construct an arbitrarily large collection of m-full words that satisfy condition (1).

We next deal with the following situation: Suppose that $\pi_1, \pi_2, ..., \pi_m$ is a terminal collection, and we concatenate an arbitrary word σ to one of the words π_i. We show that by raising all words to a high enough power we again have a terminal collection in our hands.

Lemma 1. *Let $\pi_1, \pi_2, ..., \pi_p$ be a terminal collection of full words over some alphabet. Let σ be an arbitrary full word over the same alphabet. Then the collection*

$$(\pi_1)^k, (\pi_2)^k, ..., (\pi_{i-1})^k, (\pi_i \circ \sigma)^2, (\pi_{i+1})^k, ..., (\pi_p)^k$$

is terminal as well, for every $1 \leq i \leq p$, and every $k \geq |\pi_i| + |\sigma|$.

Lemma 1 yields immediately:

Corollary 10. *Let $\pi_1, \pi_2, ..., \pi_p$ be a terminal collection of full words over some alphabet, and let $\pi_{p+1}, \pi_{p+2}, ..., \pi_n$ be arbitrary full words over the same alphabet. Then the collection*

$$(\pi_1 \circ \pi_2 \circ ... \circ \pi_n)^2, (\pi_1)^k, (\pi_2)^k, ..., (\pi_{i-1})^k, (\pi_{i+1})^k, ..., (\pi_p)^k$$

is terminal as well. This holds for every $1 \leq i \leq p$ and $k \geq \sum_{i=1}^{n} |\pi_i|$.

3.2 The MC($n, 2n-1$) Upper Bound

Our proof shows somewhat more than Theorem 1 says (see Proposition 2). We do this, since the scheduler can "trade" a player P for a chair c. Namely, he can keep P constantly on chair c. This allows the scheduler to move any other player past c-chairs. In other words this effectively means the elimination of chair c from all other words. This suggests the following definition: If π is a word over alphabet C and $B \subseteq C$, we denote by $\pi(B)$ the word obtained from π by deleting from it the letters from $C \setminus B$.

Our construction is recursive. An inductive step should add one player (i.e., a word) and two chairs. We carry out this step in two installments: In the first we add a single chair and in the second one we add a chair and a player. Both steps are accompanied by conditions that counter the above-mentioned trading option.

Proposition 2. *For every integer $n \geq 1$*

- *There exist full words $s_1, s_2, ..., s_n$ over the alphabet $\{1, 2, ..., 2n-1\}$ such that*
  $s_1(A), s_2(A), ..., s_p(A)$ is a terminal collection for every $p \leq n$ and for every subset
  $A \subseteq \{1, 2, ..., 2n-1\}$ of cardinality $|A| = 2p-1$.
- *There exist full words $w_1, w_2, ..., w_n$ over alphabet $\{1, 2..., 2n\}$, such that*
  $w_1(B), w_2(B), ..., w_p(B)$ is a terminal collection for every $p \leq n$ and for every subset
  $B \subseteq \{1, 2, ..., 2n\}$ of cardinality $|B| = 2p-1$.

The words $s_1, s_2, ..., s_n$ in Proposition 2 constitute a terminal collection and are hence an oblivious $MC(n, 2n-1)$ algorithm that proves the upper bound part of Theorem 1. The proof of Proposition 2 is given in [5].

4 The Oblivious AR$(n, 2n-1)$ Algorithm

The ideas developed to solve the musical chairs problem and prove Theorem 1 turn out to yield as well an answer to the oblivious AR problem and a proof of Theorem 2. The rules are the same as in the MC problem, except that the scheduler cannot select the initial positions, and every word is started at its first letter. In order to prove Theorem 2 we should construct a collection of full words $\Pi_N = \{s_1, s_2, ..., s_N\}$ over the alphabet $[2N-1]$ such that for every $n \leq N$ and for every set of n words from Π_N the following holds: Every schedule that starts from the first letter in each of these words reaches a safe configuration and all players only visits chairs from the set $\{1, \ldots, 2n-1\}$.

We note that our construction yields very long words - triply exponential in N. It is an interesting challenge to accomplish this with substantially shorter words.

proof (Theorem 2). By Proposition 1 and Theorem 1, we can construct for each $1 \leq i, n \leq N$ a word $\pi_{i,n}$ that is $[2n-1]$-full such that every set of n words in the set $\{\pi_{i,n} | i = 1, \ldots, N\}$ constitute an oblivious $MC(n, 2n-1)$ protocol.

We show that with a proper choice of the exponents $l_1, \ldots, l_N$, the Theorem holds with the words $s_i = \pi_{i,1}^{l_1} \circ \pi_{i,2}^{l_2} \circ \ldots \circ \pi_{i,N}^{l_N}$.

The theorem follows if we can show that for every $1 \leq n \leq N$ and every subset $J \subseteq [N]$ of cardinality $|J| = n$ the following holds: In every possible schedule that starts each word in $\{s_j | j \in J\}$ from its first letter, no player reaches a position beyond the subword $\pi_{j,n}^{l_n}$. Consider any point in such a schedule. Say that player P_j (for some $j \in J$) is *leading* if it currently resides in the stretch $\pi_{j,n}^{l_n}$ of s_j. Otherwise, we say that j is *trailing*. We observe that during a period of time in which no trailing player changes position, no leading player can traverse a complete copy of $\pi_{j,n}$. To see this, consider an arbitrary MC schedule with the words $\{\pi_{j,n} | j \in J\}$. We start this schedule as follows: Every leading player maintains his position from the original AR schedule and every trailing player stays put on the same chair that he is currently occupying. (Such a chair can be found in the word $\pi_{j,n}$ since it is $[2n-1]$-full). The claim follows since the words $\{\pi_{j,n} | j \in J\}$ constitute an oblivious $MC(n, 2n-1)$ protocol.

It follows that no leading player P_j can traverse more than $\sum_{\nu<n, i \in J \setminus \{j\}} |\pi_{i,\nu}| l_\nu$ copies of $\pi_{j,n}$ in s_j. Our claim follows if we choose l_j that is larger than this integer. □

5 Oblivious MC Algorithms by the Probabilistic Method

Remark 11. *It is important to note that the protocols that are presented in this section are* deterministic. *The constructions are, however,* inexplicit *and the existence of good protocols is proved by using a probabilistic argument. It is an intriguing open problem to find equally good explicit constructions.*

For lack of space, the full text with all proofs is given in [5]. In what follows we attempt to sketch the results.

Theorem 3 can be thought of as a (nonconstructive) derandomization of the randomized MC algorithm in which players choose their next chair at random (and future random decisions of players are not accessible to the scheduler). Standard techniques for derandomizing random processes involve taking a union bound over all possible bad events, which in our case corresponds to a union bound over all possible schedules. The asynchronous scheduler has too many options (and so does the immediate scheduler), making a union bound too wasteful. For this reason, in the analysis of this protocol we consider the canonical scheduler, which is as powerful as the asynchronous scheduler (see Section 2.3). In every unsafe configuration, a *canonical pair* of players in conflict is dictated to the canonical scheduler, and the canonical scheduler has only three possible moves to choose from. This makes it viable to use a union bound.

In this construction each of the N words is chosen independently at random as a sequence of L chairs, where each chair in the sequence is chosen independently at random. Our proof shows that with high probability (probability tending to 1 as the value of the constant c grows), this choice satisfies Theorem 3.

A simple union bound shows that in this random construction, with high probability, all words are full. Proving termination is more of a challenge. We keep track of all possible schedules. To this end we use "a logbook" that is the complete ternary tree $\mathcal{T}$ of depth L rooted at r. Associated with every node v of $\mathcal{T}$ is a random variable X_v. The values taken by X_v are system configurations. For a given choice of words and an initial system configuration we define the value of X_r to be the chosen initial configuration. Every node v has three children corresponding to the three possible next configurations that are available to the canonical scheduler at configuration X_v (and to an "empty" configuration if the scheduler cannot move). The proof uses a *potential* function that maps a configuration with i occupied chairs to x^{n-i}, where $x > 1$ is a constant optimized within the proof. In a nonempty configuration the potential is at least 1. Associated with every node of $\mathcal{T}$ is a nonnegative random variable $P = P_v$ that is the potential of the (random) configuration X_v. The main step of the proof is to show that if v_1, v_2, v_3 are the three children of v, then $\sum_{i=1}^{3} \mathbb{E}(P_{v_i}) \leq r\mathbb{E}(P_v)$ for some constant $r \leq 0.99$. This exponential drop implies that

$$\mathbb{E}\Big(\sum_{v \text{ is a leaf of } \mathcal{T}} (P_v)\Big) = \sum_{v \text{ is a leaf of } \mathcal{T}} \mathbb{E}(P_v) = o(1)$$

provided that L is large enough. This implies that with probability $1 - o(1)$ (over the choice of random words) all leaves of $\mathcal{T}$ correspond to an empty configuration. In other words every schedule terminates in fewer than L steps.

5.1 Permutations over $O(n)$ Chairs

The argument that proves Theorem 3 is inappropriate for the proof of Theorem 4. Theorem 4 deals with random permutations, whereas in the proof of Theorem 3 we use words of length $\Omega(n \log n)$. (Longer words are crucial there for two main reasons: To guarantee that words are full and to avoid wrap-around. The latter property is needed to guarantee independence.) Indeed in proving Theorem 4

our arguments are substantially different. In particular, we work with a pairwise immediate scheduler, and unlike the proof of Theorem 3, there does not appear to be any significant benefit (e.g., no significant reduction in the ratio $\frac{m}{n}$) if a canonical scheduler is used instead.

Here are some of the main ingredients of the proof of Theorem 4 for the special case $N = n$ (a slight extension of these ideas proves the general case). We show that with high probability, a set of random permutations $\pi_1, \ldots, \pi_n$ has the property that in every possible schedule the players visit at most $L = O(m \log m)$ chairs. Our analysis uses the approach of deferring random decisions until they are actually needed. For each of the m^n possible initial configuration, we consider all possible sequences of L locations. For each such sequence we fill in the chairs in the locations in the sequence at random, and prove that the probability that this sequence represents a possible schedule is extremely small – so small that even if we take a union bound over all initial configurations and over all sequences of length L, we are left with a probability much smaller than 1.

The main difficulty in the proof is that since $L \gg m$ some players may completely traverse their permutation (even more than once) and therefore the chairs in these locations are no longer random. To address this, we partition the sequence of moves into L/t blocks, where in each block players visit a total of $t = \delta m$ locations for some sufficiently small constant δ. Also $n = \epsilon m$, where ϵ is a constant much smaller than δ. This choice of parameters implies that within a block, chairs are essentially random and independent. To deal with dependencies among different blocks, we classify players (and their corresponding permutations) as *light* or *heavy*. A player is *light* if during the whole schedule (of length L) it visits at most $t/\log m = o(t)$ locations. A player that visits more than $t/\log m$ locations during the whole sequence is *heavy*. For light players, the probability of encountering a particular chair in some given location is at most $\frac{1}{m-o(t)} \leq \frac{1+o(1)}{m}$. Hence, the chairs encountered by light players are essentially random and independent (up to negligible error terms). Thus it is the heavy players that introduce dependencies among blocks. Every heavy player visits at least $t/\log m$ locations. Hence the number n_h of heavy players does not exceed $(L \log m)/t = O(\log^2 m)$. The fact that the number of heavy players is small is used in our proof to limit the dependencies among blocks.

6 Open Problems

Our MC algorithms involve very long words. An interesting question is to find explicit constructions with $m = 2n - 1$ chairs and substantially shorter words.

In other ranges of the problem we can show, using the probabilistic method, that oblivious $MC(n, m)$ algorithms exist with $m = O(n)$ and relatively short full words. We still do not have explicit constructions of such protocols. We would also like to determine $\liminf \frac{m}{n}$ such that n random words over an m letter alphabet tend to constitute an oblivious $MC(n, m)$ algorithm.

Computer simulations strongly suggest that for random permutations, a value of $m = 2n - 1$ does not suffice. On the other hand, we have constructed (details omitted from this manuscript) oblivious $MC(n, 2n - 1)$ algorithms using permutations for $n = 3$ and $n = 4$ (for the latter the proof of correctness is computer-assisted). For $n \geq 5$ we have neither been able to find such systems (not even in a fairly extensive computer search) nor to rule out their existence.

A self contained proof of the $m \geq 2n - 1$ lower bound will appear in a subsequent paper. The following question remains open: What is the smallest m for which there are collections of $N = m + 1$ (not necessarily full) words such that every $\min[n, N]$ of them form an oblivious MC algorithm *when starting at the initial chair of each word*. Our proof that $m \geq 2n-1$ assumes that the scheduler is allowed to pick an arbitrary initial state on each word.

We do not know how hard it is to recognize whether a given collection of words constitute an oblivious MC algorithm. This can be viewed as the problem whether some digraph contains a directed cycle or not. The point is that the digraph is presented in a very compact form. It is not hard to place this problem in PSPACE, but is it in a lower complexity class, such as co-NP or P?

There are interesting foundational questions related to different models in distributed computing. We have defined here the Output Negotiation (ON) model, and showed that it is properly included in the read/write model. It follows by definition that the oblivious model is included in the ON model. It would be interesting to know whether this last inclusion is proper.

Acknowledgements. Work of Uriel Feige was supported in part by The Israel Science Foundation (grant No. 873/08). Work of Benny Sudakov was supported in part by NSF grant DMS-1101185, NSF CAREER award DMS-0812005 and by USA-Israeli BSF grant.

References

1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems. In: FOCS, pp. 218–223 (1979)
2. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM 37(3), 524–548 (1990)
3. Anderson, R.J., Woll, H.: Algorithms for the Certified Write-All Problem. SIAM J. Comput. 26(5), 1277–1283 (1997)
4. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic Snapshots of Shared Memory. Journal of the ACM 40(4), 873–890 (1993)
5. Afek, Y., Babichenko, Y., Feige, U., Gafni, E., Linial, N., Sudakov, B.: Oblivious Collaboration (ArXiv version of current paper.), `http://arxiv.org/abs/1106.2065`
6. Afek, Y., Alon, N., Bar-Joseph, Z., Cornejo, A., Haeupler, B., Kuhn, F.: Beeping a Maximal Independent Set. In: Proc. 25th Int'l Symposium on Distributed Computing (DISC 2011), Rome Italy (September 20-22, 2011)

7. Afek, Y., Gafni, E., Rajsbaum, S., Raynal, M., Travers, C.: Simultaneous consensus tasks: A tighter characterization of set-consensus. In: Chaudhuri, S., Das, S.R., Paul, H.S., Tirthapura, S. (eds.) ICDCN 2006. LNCS, vol. 4308, pp. 331–341. Springer, Heidelberg (2006)
8. Beame, P., Blais, E., Ngoc, D.: Longest common subsequences in sets of permutations, http://arxiv4.library.cornell.edu/abs/0904.1615?context=math
9. Borowsky, E., Gafni, E.: Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. In: Proc. 25th ACM Symposium on Theory of Computing (STOC 1993), pp. 91–100 (1993)
10. Castañeda, A., Rajsbaum, S.: New combinatorial topology upper and lower bounds for renaming. In: PODC, pp. 295–304 (2008)
11. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM 17(11), 643–644 (1974)
12. Dolev, S.: Self-Stabilization. MIT Press, Cambridge, ISBN 0-262-04178-2
13. Dolev, D., Lynch, N.A., Pinter, S., Stark, E.W., Weihl, W.E.: Reaching Approximate Agreement in the Presence of Faults. In: Symposium on Reliability in Distributed Software and Database Systems, pp. 145–154 (1983)
14. Gafni, E.: Read-write reductions. In: Chaudhuri, S., Das, S.R., Paul, H.S., Tirthapura, S. (eds.) ICDCN 2006. LNCS, vol. 4308, pp. 349–354. Springer, Heidelberg (2006)
15. Gafni, E., Mostéfaoui, A., Raynal, M., Travers, C.: From adaptive renaming to set agreement. Theor. Comput. Sci. 410(14), 1328–1335 (2009)
16. Gafni, E., Rajsbaum, S.: Musical benches. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 63–77. Springer, Heidelberg (2005)
17. Gafni, E., Rajsbaum, R., Raynal, M., Travers, C.: The committee decision problem. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 502–514. Springer, Heidelberg (2006)
18. Herlihy, M.P., Shavit, N.: The Topological Structure of Asynchronous Computability. Journal of the ACM 46(6), 858–923 (1999)
19. Metcalfe, R.M., Boggs, D.R.: Ethernet: Distributed packet switching for local computer networks. Commun. Ass. Comput. Mach. 19(7), 395–404 (1976)
20. Saks, M., Zaharoglou, F.: Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. SIAM Journal on Computing 29(5), 1449–1483 (2000)