Bounds on the shared memory requirements for Long-Lived & Adaptive objects

[Extended Abstract]

Yehuda Afek
Computer Science Dept.
Tel-Aviv University and IDC
Herzliya
Israel 69978
afek@math.tau.ac.il

Pazi Boxer
Computer Science Dept.
Tel-Aviv University
Israel 69978
pazi_boxer@starhome.com

Dan Touitou
IBM Research Lab in Haifa
Israel
dant@il.ibm.com

ABSTRACT

In this paper we prove:
For any constant d there is a large enough n such that there is no long-lived adaptive implementation of collect or renaming in the read write model with n processes that uses d or less MWMR registers.

In other words, there is no implementation of a long-lived and adaptive renaming or collect object in the atomic read/write model that uses O(1) multi-writer-multi-reader registers and any number of single-writer-multi-reader registers. In 1980 Burns and Lynch [1] proved that at least n multi-writer-multi-reader (MWMR) registers are necessary in any mutual exclusion algorithm that uses only MWMR registers (i.e., atomic registers). It is also relatively easy to see that any adaptive non-trivial algorithm uses at least one multi-writer-multi-reader (MWMR) register even when there are n single-writer-multi-reader (SWMR) registers. Here we extend the techniques of Burns and Lynch and prove that adaptive algorithms that use both SWMR and MWMR registers such as, collect and renaming, need in addition to the $\Omega(n)$ SWMR registers a non-constant, $F(n)$ number of MWMR registers.

1. INTRODUCTION

In recent years, several different long-lived adaptive algorithms whose operation step complexity is a function of the actual number of processes running concurrently with the operation execution, have been presented [2, 3, 4]. Regardless of the type of adaptiveness and which object was implemented, they all require $\Omega(n)$ MWMR registers in their implementation, where n is the total number of processes that *may* participate in the algorithm/implementation. Motivated by the question whether n MWMR registers are a necessity we prove here that the number of MWMR registers used by either long-lived adaptive renaming or collect must be a function of n. Specifically, for any constant d there is a large enough n_d such that every long-lived adaptive read/write implementation of collect (and renaming) with n_d processes must use at least d MWMR registers.

The main proof in this paper is given for a simple object called *weak test and set* [5], or WT&S in short. Below we give long-lived and adaptive reductions from collect and adaptive renaming objects to WT&S, thus proving our main theorem. A weak test and set object (fully specified in Subsection 2.1) behaves like a test-and-set object except that if a process tries to acquire the test-and-set bit and at least one more process is running concurrently with it, then it may fail. That is, if several processes invoke a weak-test-and-set operation concurrently, they may *all* fail, leaving the test-and-set bit vacant, un-captured. Proving the main result for a simple and weak object like WT&S enables us to reduce the result to other objects, as we show below for renaming and collect.

In proving the impossibility of an adaptive long-lived WT&S object, we follow the Burns and Lynch proof method which is extended in a non-trivial way here. The proof constructs a run in which two processes p and q win the test-and-set bit. At some point in the constructed run process p captures the weak-test-and-set bit, then a set of processes write on all the MWMR registers written by p erasing all traces of p. This enables another process q to acquire the bit, violating the exclusion property of weak-test-and-set.

The operational semantics (specification) of a collect object is as follows: there are n single-writer-multi-reader registers $C_1, \ldots, C_n$, one for each process. To write in the collect object process P_i writes its value in C_i. To perform a collect operation a concurrent process simply reads the n registers and returns the vector of values read.

Reduction from collect to WT&S:

To implement a WT&S object from an adaptive collect object a process simply writes "active" in the collect object and performs an (adaptive) collect operation (Initially all the values in the collect object are "asleep"). If it observes only itself with an "active" value in the collected set, it wins the test and set bit. Otherwise it writes "asleep" and fails in the WT&S operation. A winning process later releases the test and set bit by writing "asleep" in the collect object.

Reduction from adaptive renaming to WT&S:
In the *long-lived adaptive renaming* problem, processes repeatedly acquire and release distinct names; A name acquired by a process must be in the range $\{1,\ldots,f(k)\}$, where k is the number of processes that hold names or are actively acquiring a name concurrently with the process that acquires a new name. In particular, for any adaptive renaming the size of a name acquired when a process runs solo (i.e., when no process holds a name or tries to acquire a name) is bounded by some constant $f(1)$. In a long lived and adaptive WT&S implementation from adaptive renaming we use one copy of the adaptive renaming object and an array of $f(1)$ Boolean MWMR registers. A process performs the WT&S operation by first acquiring a name and checking if the name size is smaller or equal to $f(1)$. If the name size is larger the process has failed the WT&S and releases the name it has acquired. Otherwise, it verifies that it is the only process holding a name smaller or equal to $f(1)$. To perform this verification it writes "active" in the MWMR register that corresponds to the name it got, and reads all the $f(1)$ MWMR registers. If all other registers are read "asleep" the process safely wins the WT&S operation. To reset the weak-test-and-set bit a process writes "asleep" in the corresponding MWMR register, and releases the name it had acquired.

Notice that obviously there is no adaptive implementation of a multi-writer-multi-reader register from a system of single-writer-multi-reader registers. This follows because such an implementation uses only single-writer registers. Hence, to read a value written into the implemented multi-writer register a process would have to read all of them or otherwise it would miss the last value written.

Notice that the long-lived and adaptive implementation of collect in [3] uses $O(n)$ MWMR registers, thus leaving a gap between the linear upper and the non-constant lower bound on the number of MWMR registers necessary for their adaptive implementation of a collect object.

Farther notice that there is an adaptive *one-shot* ϵ-agreement algorithm that uses only a constant number of shared (MWMR) registers [6]. The number of shared registers in that algorithm is inversely proportional to ϵ.

The formal proof of the main theorem is given in [7]. In the body of the paper, Section 2 we make an attempt to describe the proof in a non-formal way. In Subsection 2.1 we define the *weak test and set* object. In Subsection 2.2 we state the theorem, and in Subsection 2.3 we intuitively provide the definitions necessary for the informal description of the proof. In Subsection 2.4 the grand plan of the proof is described, while the informal description of the proof of the main lemma is given in Subsection 2.5. Concluding remarks are given in Section 3.

2. INFORMAL DESCRIPTION OF THE THEOREM AND ITS PROOF

2.1 WT&S

The behavior of a WT&S object is modeled by the following program. Each process is in one of four possible states in this program: thinking, WT&SET, eating and RESET.

```
TS: object of type WT&S
Process p:
repeat forever
      thinking_section_p
      tbit := WT&SET_p(TS)
      if tbit=0 {
            eating_section_p
            RESET_p(TS) }
end repeat forever
```

A WT&S object satisfies the following two properties:

Exclusion: At most one process is eating at any point (system-state) of the execution.

Solo: If a process becomes hungry (goes out of the thinking state), while all other processes are thinking, and runs alone (it is the only process that takes steps) then it must eventually start eating.

2.2 The Theorem

Clearly, if an implementation of an object is adaptive, then there is a constant k such that, no solo run segment has more than k steps. I.e., in any solo run of a process (all other processes are thinking) the implemented WT&SET and RESET operations finish within k primitive operations. We say that the algorithm is *"k-solo-adaptive"*.

THEOREM 2.1. *For any constants d and k, there is a large enough number $n_{d,k}$ s.t., there is no long-lived k-solo-adaptive WT&S implementation in a system with $n_{d,k}$ or more processes, and with d or less Multi-Writer Multi-Reader registers.*

COROLLARY 2.2. *For any constant d, there is a large enough n_d s.t., there is no long-lived adaptive WT&S implementation in a system with n_d processes, that uses d or less Multi-Writer Multi-Reader registers.*

Proof of Corollary 2.2: By contradiction, assume that there is an adaptive implementation of WT&S object that uses d MWMR registers and works for any number of processes. Then consider one such implementation, and let k be its k-solo-adaptive parameter. By theorem 2.1, for $n > n_{d,k}$ no such implementation exists, a contradiction. □

The following corollary follows from Corollary 2.2 using the reductions from Renaming and Collect objects to WT&S object, presented in the introduction.

COROLLARY 2.3. *For any constant d, there is a large enough n'_d s.t., there is no long-lived adaptive renaming or collect objects implementation in a system with n'_d processes, that uses d or less Multi-Writer Multi-Reader registers.*

Henceforth, the rest of this extended abstract overviews the proof of Theorem 2.1.

2.3 Preliminaries and Definitions

- A *system state* consists of the state of all processes and the value of all registers in the system. A system has one or more initial system states in which the system starts its execution.
- We say that a system state s is an *idle-state*, if all the processes are thinking at s.
- A *run* α is a finite or infinite sequence of events, starting from an initial system state. When the sequence is finite, we also say that the run is finite.
- A *run segment* x_α of run α is a finite, continuous subsequence of events of α.
- A *solo run segment* of p is a run segment starting in an idle-state, in which only p takes steps.
- A *p-segment* is a run segment starting in a state s in which process p is thinking, and in which only p takes steps. Notice that a p-segment is not necessarily a p solo segment.
- A state s is *transparent* with respect to process p, if p is thinking in s, and there is a p-segment starting in s, that p cannot distinguish from some solo run segment of p starting at an idle-state and ending with p eating.

  Similarly, state s is transparent with respect to a set of processes Q, if s is transparent $\forall p \in Q$.
- A *pseudo-solo run segment* of p, denoted $^\sim solo_p$, is a p-segment starting in state s s.t., s is transparent with respect to p, and ending with p eating. (By the definition of transparent, p cannot distinguish the pseudo-solo run segment from some solo run segment of p starting at an idle-state and ending with p eating.)
- We say that a register r is *covered* by process p at the end of run α, if a write operation by p to r is enabled at the end of α.
- Given a run segment x and a state s, $s \cdot x$ denotes the concatenation of x after some run α which ends at s, assuming that such an α exists.

A WT&S implementation is *adaptive* [8, 9], if the number of primitive operations (i.e., read/write operations) is bounded by a function of the contention during the high-level operation (either WT&SET or RESET) execution. Since we are proving an impossibility result, the weaker notion of contention gives us a stronger result. Therefore, we consider the contention to be the *interval* contention [8, 9], which is the total number of processes that are active during the time interval of the high-level operation execution.

W.l.o.g, we assume that each process has only one single-writer multi-reader (SWMR) register. All the SWMR registers of a process can always be replaced by one large SWMR register.

We say that a condition or property holds in a run, if it holds at the end of the run (unless we specifically say otherwise).

2.4 Sketch of Proof of the theorem

The proof is by a way of contradiction. We first assume that there is a WT&S implementation with d MWMR registers for a system with more than $n_{d,k}$ processes. Then, we show that under these conditions there is a run in which two processes, p and q are in the critical section (i.e., eating), at the same time. There are 6 steps in the construction of this run, where by far, the most difficult step is the first step: (See drawing 1, which illustrates this construction.)

1. We construct a run prefix α' s.t., the state at the end of α' is transparent with respect to p, and every MWMR register that p writes in its pseudo-solo run starting at α', is covered. (See step 1 in Figure 1.)
2. Let $^\sim solo_p$ be the pseudo-solo run segment of process p, starting after α'. Hence p is eating in $\alpha' \cdot^\sim solo_p$. Let $\{r_1, ..., r_{k'}\}$ be the set of MWMR registers written by p in $^\sim solo_p$, where $k' \le k$. (See step 2 in Figure 1.)
3. We continue the construction with a run segment *clean* by enabling the covering writes in the following way: $\forall r_i \ i = 1, ..., k'$, we let the process u_i that covers r_i at $\alpha' \cdot^\sim solo_p$ write. After all registers in $\{r_1, ..., r_{k'}\}$ are covered, let each process u_i continue its execution until it reaches the thinking state. I.e., in $\alpha' \cdot^\sim solo_p \cdot clean$ all the MWMR registers written by p have been overwritten. (See steps 3 and 4 in figure 1.)
4. In the full proof in [7], both the set $\{u_i\}\ i = 1, ..., k'$ and process p are carefully selected in such a way that none of the processes u_i reads the single-writer multi-reader register R_{SW_p} that p writes during $^\sim solo_p$. Thus, all the MWMR registers that p writes are overwritten and the only trace of p, the content of its SWMR register, is ignored by processes $u_1, ..., u_{k'}$. Therefore, in $\alpha' \cdot^\sim solo_p \cdot clean$ all the MWMR registers written by p have been overwritten, and have the same values that they have in some idle state. (See steps 3 and 4 in figure 1.)
5. We show that there are processes p and q s.t., run $\alpha' \cdot^\sim solo_p \cdot clean$ is transparent with respect to q. This is done by carefully selecting processes p and q in such a way that q does not read the SWMR register that p writes in $^\sim solo_p$. Therefore, there is a pseudo-solo run segment $^\sim solo_q$ starting after $\alpha' \cdot^\sim solo_p \cdot clean$ in which q is eating. (See step 5 in figure 1.)
6. At the end of $\alpha' \cdot^\sim solo_p \cdot clean \cdot^\sim solo_q$, both p and q are in the critical section, a contradiction.

In the full proof in [7], we show that the size of $n_{d,k}$ is a function of d, and k and these constants may be different for different implementations.

The sketch of proof for the inductive construction of run α', is described in the next section.

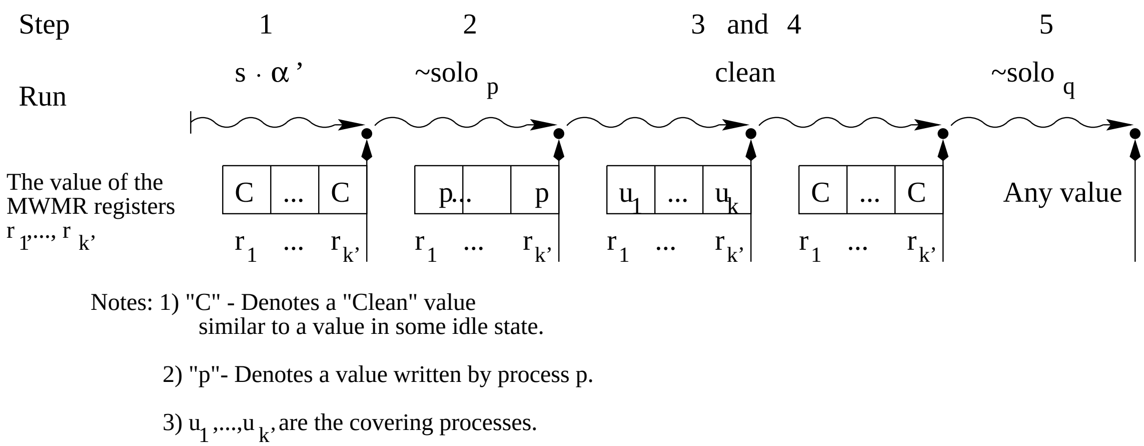


Figure 1: Illustration of the proof of Theorem 2.1.

2.5 Sketch of Proof of the Main Lemma

In the main lemma we construct the required run α', by using several techniques.

For explanatory reasons only and only in this section, we separate these techniques by defining assumptions, starting with very strong assumptions, which are later relaxed until removing all the assumptions. This is done in three phases, where in each phase we define the lemma again in an informal manner (see lemma's 2.4, 2.5, 2.6 and 2.7). The formal proof of the lemma without any assumption is provided in [7].

Henceforth in the rest of this paper, we use the following notations. For any constant d, we consider W to be an adaptive implementation of WT&S in the Read/Write shared memory model, with a set of d multi-writer multi-reader registers, R_{MW}. Let k be the length of the longest solo run segment i.e., W is k-solo-adaptive. Notice that by the definition of adaptiveness, k is a constant.

Phase 1:

Assumption A: There are no write operations to SWMR registers in all legal runs. That is, we assume for the moment that there is a wait free algorithm that uses only d MWMR and no SWMR registers.

Assumption B: If G is a set of processes and s is a state which is transparent with respect to G, then during their pseudo-solo run starting at s all the processes in G write in the same MWMR registers at the same order.

Although, under assumption A, the impossibility result presented in this paper follows directly from Burns & Lynch [1], we still provide it here, since the proof serves as the basis for our final proof.

In the following lemma, we construct α', as in step 1 in section 2.4. In the lemma, run α' is denoted by $s \cdot \beta$ and it satisfies the properties of α'. Property (1): the state at the end of run $s \cdot \beta$ is transparent with respect to some set of processes called G_e, and property (2): there is a cover on all the MWMR registers written by processes in G_e in their pseudo-solo run segments, starting after $s \cdot \beta$. The size of the set G_e is a parameter and it is determined in the proof of theorem 2.1.

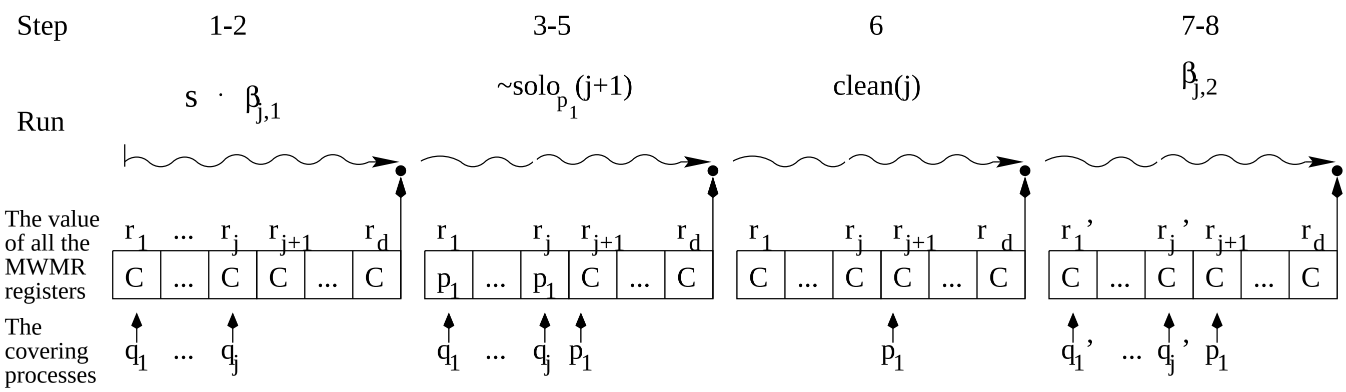
Lemma 2.4. *Let W be a long-lived k-solo-adaptive WT&S algorithm satisfying assumptions A and B. Then for any constant e there exists a constant n_e s.t., for any set of processes G, $|G| \geq n_e$ and for any state s transparent with respect to G, there is a run segment β and a set of processes $G_e \subseteq G$ s.t., the following holds: (1) $|G_e| \geq e$, (2) the state at the end of $s \cdot \beta$ is transparent with respect to G_e, and (3) all the MWMR registers written in the pseudo-solo run segments of processes in G_e, after $s \cdot \beta$, are covered in $s \cdot \beta$.*

Sketch of proof of Lemma 2.4: The proof is by induction on i, the inductive claim is:

Lemma 2.5. *Let W be a long-lived k-solo-adaptive WT&S algorithm satisfying assumptions A and B. Then for every i, $0 \leq i \leq k$ and for every constant e there exists a constant $n_{e,i}$ s.t., for any set of processes G, $|G| \geq n_{e,i}$ and for any state s transparent with respect to G, there is a run segment β_i and a set of processes $G_i \subseteq G$ s.t., the following holds: (1) $|G_i| \geq e$, (2) the state at the end of run $s \cdot \beta_i$ is transparent with respect to G_i, and (3) either the first i MWMR registers written in the pseudo-solo run segments of processes in G_i, after $s \cdot \beta_i$, are covered in $s \cdot \beta_i$, or, the pseudo solo runs starting at $s \cdot \beta_i$ have less than i writes to MWMR registers and all these writes are covered in $s \cdot \beta_i$.*

Sketch of proof of Lemma 2.5 : (See drawing 2, which illustrates the construction of α' until step 8.) In the full proof in [7], we show that $n_e = n_{e,i}$ is a function of d, i and e.

1. For $i = 0$ the claim holds trivially by setting $G_0 = G$, $n_{e,0} = e$ and β_0 to be the empty sequence.



Notes: 1) "C" - Denotes a "Clean" value, similar to a value in some idle state.

2) "p" - Denotes a value written by p.

3) An arraw means that the process is covering the register.

Figure 2: Illustration of the memory state in proof of Lemma 2.5 steps 1- 8 (run $\alpha_{j+1,1}$).

2. We assume that the claim holds for $i = j$ and we show that it holds for $i = j+1$. Assume that G is a large enough set of processes s.t., $|G| \geq n_{e,j+1}$ and s is a system state transparent with respect to G. In the end of the proof we show how to calculate $n_{e,j+1}$. By the induction hypothesis, there exist $G_{j,1}$ and $\beta_{j,1}$ such that at the end of $s \cdot \beta_{j,1}$, writes to the first j MWMR registers written in the pseudo-solo run segments of processes in $G_{j,1}$, are enabled and $s \cdot \beta_{j,1}$ is transparent with respect to $G_{j,1}$. By assumption B, let $(r_1, ..., r_j)$ be the first j MWMR registers written in the pseudo-solo run segments of processes in $G_{j,1}$. (See steps 1-2 in drawing 2.)

3. Select any process $p_1 \in G_{j,1}$ and a pseudo-solo run segment $^{\sim}solo_{p_1}$ starting at $s \cdot \beta_{j,1}$.

4. If there are less than $j+1$ write operations to MWMR registers in $^{\sim}solo_{p_1}$ (until p_1 eats), then we are done and $\beta_i = \beta_{j,1}$ for $i = j, j+1, \ldots k$ (I.e., if p_1 does not have more than j write operations to MWMR registers, then none of the processes in $G_{j,1}$ has a $j+1$st write to a MWMR register, since, by Assumption B, they all write in the same MWMR registers in the same order, until eating).

5. Otherwise, p_1 has at least $j+1$ write operations in MWMR registers in $^{\sim}solo_{p_1}$, before eating. Let operation (event) $W_{p_1,j+1}$ be the $j+1st$ write to a MWMR register by process p_1. Let r_{j+1} be the register that process p_1 writes in its $W_{p_1,j+1}$, and let $^{\sim}solo_{p_1}(j+1)$ be the longest prefix of $^{\sim}solo_{p_1}$ that does not include $W_{p_1,j+1}$. Any pseudo-solo run segment that starts after $s \cdot \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1)$ can read p_1's write operations to $r_1, ..., r_j$ and therefore, the state is not transparent. (Notice that because of assumption B, p_1 is enabled to write in the same MWMR register r_{j+1} as all processes in $G_{j,1}$ in their pseudo-solo run segments, starting at $s \cdot \beta_{j,1}$.) (See steps 3-5 in drawing 2.)

6. To erase all the traces of p_1, we next enable each of the processes $q_1, ..., q_j$ which are covering $r_1, ..., r_j$ respectively to write in $r_1, \ldots, r_j$ and to run until they reach a thinking state. We denote this run segment as $clean(j)$. Such a $clean(j)$ extension to $s \cdot \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1)$ exists, since W is a wait-free WT&S implementation. Therefore, each process $q_1, ..., q_j$ reaches its thinking state after a finite number of steps. Let $\alpha_{j+1,1} = \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1) \cdot clean(j)$. Then, by the end of run $s \cdot \alpha_{j+1,1}$ the state is transparent with respect to $G_{j,1} - \{p_1\}$. Here we use assumption A, which permits write operations only in MWMR registers, otherwise, later pseudo solo executions could observe p_1 by reading its SWMR register. By the end of run $s \cdot \alpha_{j+1,1}$, only one process, p_1, is enabled to write in r_{j+1}, while the registers $r_1, ..., r_j$ are not necessarily covered any more. (See step 6 in drawing 2.)

7. Since the state at $s \cdot \alpha_{j+1,1}$ is transparent with respect to $G_{j,1} - \{p_1\}$, we can activate the induction hypothesis again. Hence a cover of a sequence of j MWMR registers is constructed by first activating (by the induction) a run segment $\beta_{j,2}$ s.t., the state at the end of $s \cdot \alpha_{j+1,1} \cdot \beta_{j,2}$ is transparent to a set of processes $G_{j,2} \subseteq G_{j,1} - \{p_1\}$. Let $r'_1, ..., r'_j$ be the new j MWMR registers covered at the end of $s \cdot \alpha_{j,1} \cdot \beta_{j,2}$ and let $q'_1, ..., q'_j$ be the processes covering them. (See steps 7-8 in drawing 2).

REPEAT-COVER technique:

8. Let $r'_1, r'_2, ..., r'_j, r'_{j+1}$ be the first $j+1$ registers written in the pseudo-solo segments of the processes in $G_{j,2}$. Assume that $r'_{j+1} = r_{j+1}$. In such a case, we are done, since $r'_1, r'_2, ..., r'_j$ are covered by processes $q'_1, ..., q'_j$ and r_{j+1} is covered by process p_1. (See steps 7-8 in drawing 2).

9. If however $r'_{j+1} \neq r_{j+1}$, (because the pseudo solo runs in $\beta_{j,2}$ do not necessarily write to the same registers as the pseudo solo runs in $\beta_{j,1}$) then we construct the required cover on r'_{j+1}, by repeating steps 3- 7 above, but starting at $s \cdot \alpha_{j+1,1} \cdot \beta_{j,2}$. I.e., we keep process p_1 covering r_{j+1} and extend the run with a segment in which a process $p_2 \in G_{j+1,2}$ is covering r'_{j+1}. After repeating steps 3- 7 once, we have a cover on both

r_{j+1} and r'_{j+1}, by two different processes, p_1 and p_2 in the run $s \cdot \alpha_{j+1,1} \cdot \alpha_{j+1,2} \cdot \beta_{j,3}$. We repeat and construct a new cover on another $j+1st$ register, until either one of the following: we finish in step 4, or in step 8, or we repeat until all MWMR registers are covered, i.e., up to at most d times.

By the termination condition of this loop, there is now a cover on all $j+1$ MWMR registers written by the current collection of pseudo solo runs. Since d is a constant, the run we have constructed is of constant length and there are enough processes to construct it with the repeating covers, since G is a large enough set.

10. In conclusion for $i = j+1$, $\beta_i = \beta_{j+1,d} = s \cdot \alpha_{j+1,1} \cdot \cdot \cdot \alpha_{j+1,d} \cdot \beta_{j,d+1}$ the concatenation of all repeating run segments in steps 3- 7 and $G_{j+1} = G_{j,d+1}$ and $|G_{j+1}| \geq e$, (which is the set G after removing from it the processes participating in $\beta_{j+1,d}$).

We can now estimate $n_{e,j+1}$, the minimum number of processes necessary to construct β_{j+1} such that in the end of $s \cdot \beta_{j+1}$ there will be at least e processes in G_{j+1}. The inductive step going from j to $j+1$ requires at most $d+1$ applications of the induction hypothesis. We therefore define $n_{e,j+1}$ with the help of an auxiliary function f_e inductively as follows: $n_{e,j+1} = f_e(j,d)$, $f_e(0,x) = e$, $f_e(x,0) = e$ and if $j \neq 0$ then $f_e(j,x+1) = n_{f_e(j,x)+1,j}$. Clearly, if e and d are constants then $n_{e,j+1}$ is a constant, too.

□

Phase 2:

Here assumption A is relaxed i.e., during the WT&S operations processes may write in their SWMR registers. The run constructed in the proof of Lemma 2.5, may not be valid anymore, as processes are allowed to write in their SWMR registers. Consider the case in the run in which a SWMR register of an active (i.e., not thinking) process p, is read by another process q. Thus, process q can point out at p (e.g., by leaving in the MWMR registers a "look at p's SWMR register" instruction). Therefore, this run cannot reach a transparent state as long as p is active. The reasoning presented in step 6 may collapse in one of the following two ways:

(1) The participating processes in any *clean* run segment may read the SWMR registers of other active processes, like process p. By this, leaving the system in a non-transparent state with respect to none of the processes. For more details see case 1 in the sketch of proof, below.

(2) After some *clean* run segment, a process q starts its q-segment execution and it may read a SWMR register of another active process p. Thus, q does not perform a pseudo-solo run segment any more, instead it may stop its execution without covering the MWMR registers. In addition q may write in different MWMR registers, because assumption B is defined only for pseudo-solo run segments. For more details see case 2 in the sketch of proof, below.

These problematic situations are avoided here by preventing from processes whose SWMR registers are later read from taking part in the constructed run. Therefore, in any given state in the run (of phase one), if process q reads the SWMR register of process p and p is active, then we construct another run in which p is replaced by another process p'. This replacement is such that process q still reads the same SWMR registers, i.e., the register of p and not of p'. Let β be the run in which p participates and let β' be a new run in which another process p' is participating instead of p. Process p''s behavior in β' is somehow "equivalent" to p's behavior. That is p and p' are writing and covering the same MWMR registers in β and β', respectively. We show that a process like p' always exists because: (1) There is a large enough set of processes to select p' from s.t., p' did not participate in the run before. This is because only a function of d and k processes are participating in the run constructed in phase one. (2) Process q can perform only a constant number of read operations, since the number of concurrently active processes in the run is a function of d and k and since the algorithm is adaptive.

During the inductive proof below, we maintain a large enough set of "equivalent" runs, which allow us to replace the run at any point in which we fail to reach a transparent state. This set is reduced as the construction of the run progresses.

Definition 2.1. *Two runs β and β' are equivalent with respect to a set of processes G if (1) the state at the end of both runs β and β' is transparent with respect to G, (2) the sets of MWMR registers covered in β and β' are the same, and (3) if process p participates in both β and β' then p cannot distinguish between the two runs.*

While constructing a run in the proof, whenever a process p which had been selected to participate in the run is discovered by covering processes, we need to replace it with some other process p' that cannot be discovered. This is achieved by considering an equivalent run in which p' takes steps instead of p. We therefore restate inductive lemma 2.5 as follows:

Lemma 2.6. *Let W be a long-lived k-solo-adaptive WT&S algorithm satisfying assumption B. Then for every i, $0 \leq i \leq k$ and for every constant e there exists a constant $n_{e,i}$ s.t., for any set of processes G, $|G| \geq n_{e,i}$ and for any state s transparent with respect to G, there is a run segment β_i and a set of processes $G_i \subseteq G$ s.t., the following holds: (1) $|G_i| \geq e$, (2) the state at the end of run $s \cdot \beta_i$ is transparent with respect to G_i, and (3) either the first i MWMR registers written in the pseudo-solo run segments of processes in G_i, after $s \cdot \beta_i$, are covered in $s \cdot \beta_i$, or, the pseudo solo runs starting at $s \cdot \beta_i$ have less than i writes to MWMR registers and all these writes are covered in $s \cdot \beta_i$. And in addition, there is a large set of runs* equivalent *to β_i with respect to a large enough set $G'_i \subseteq G_i$.*

Notice that the proof of the theorem given the above lemma should be also modified. The modified proof of Theorem 2.1 follows the same lines as the proof of Lemma 2.6 and is not included in this extended abstract.

Sketch of proof of Lemma 2.6: A set of runs equivalent to β_{j+1} is constructed s.t., each run is equivalent with respect to a sub-set of the set of processes, G_{j+1}. Unless stated otherwise we use the same notations and symbols as in phase one. We do not provide here an evaluation of the size of $n_{e,j+1}$. We only go through the modifications necessary to the steps of phase one.

Equivalent set of runs:

As explained above, in step 6, since assumption A is removed the following two cases should be addressed:

1. After run $s \cdot \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1) \cdot clean(j)$, if any of the processes which participates in $clean(j)$ (i.e., $q_1, ..., q_j$), reads from the SWMR register $R_{SW_{p_1}}$, which p_1 wrote, then the state is not transparent with respect to $G_{j,1}$. However, since W is an adaptive algorithm, the number of steps in $clean(j)$ is a constant, given that k and d are constants. Therefore, the total number of SWMR registers that may be read by any $q_1, ..., q_j$ during any $clean(j)$ run segment is bounded by some constant, say l. Hence, because of the one-to-one correspondence between processes and SWMR registers, at most l active processes may be discovered during $clean(j)$ run segment. Removing these l processes from $G_{j,1}$ leaves us with a large enough set $G_{invis} \subseteq G_{j,1}$ ($invis$ stands for invisible).

   Unfortunately process p_1 is not the only process that may be found active by $q_1, ..., q_j$ during $clean(j)$. In fact, since we repeat the inductive construction up to d times in step 9 of phase one, $clean(j)$ may be a part of the d'th repetition. In such a case, in addition to p_1 all the other $d-1$ processes that have been chosen to participate in the $j+1$'st cover are alive and may be discovered by $q_1, ..., q_j$ during $clean(j)$, as well.

   Consider the following example of the above situation: (For an illustration of the corresponding memory snapshot see figure 3).

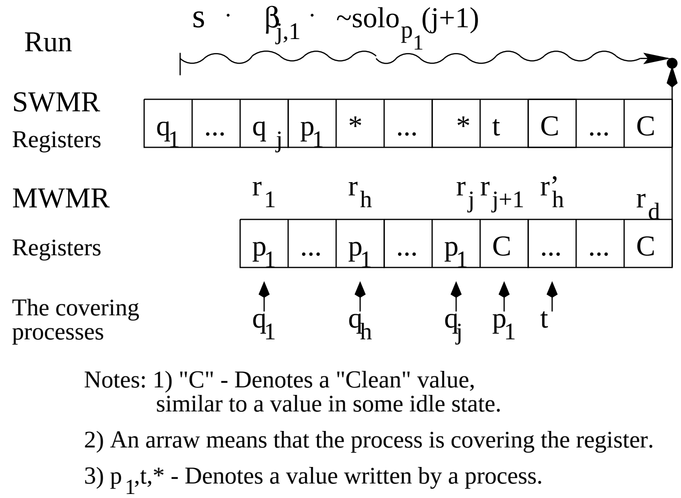


Figure 3: Illustration of the memory snapshot, after run $s \cdot \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1)$ during step 6 in proof of Lemma 2.6.

   Suppose processes $q_1, ..., q_j$ read R_{SW_t} the SWMR register of an active process t. If in some induction step $h < j+1$, t was selected to cover the MWMR register r'_h, where $r'_h \neq r_h$, then obviously the induction step was repeated at least one more time in order to cover r_h by q_h. Therefore, both t and q_h are active at the end of $s \cdot \beta_{j,1}$.

   To overcome this problem, we need to construct a run $s \cdot \beta^*_{j,1}$ equivalent to $s \cdot \beta_{j,1}$ with respect to some set of processes, in which a process like t is "replaced" by another process t' s.t., none of processes $q_1, ..., q_j$ reads the SWMR register of t'.

   Intuitively, constructing a new run equivalent to run $s \cdot \beta_{j,1} \cdot^{\sim} solo_{p_1}(j+1)$, is done by repeating the construction of the current run from the beginning i.e., $i = 0, ..., j$. In the new construction, we remove active processes that are later "discovered" by $q_1, ..., q_j$. In their place we select other processes and their pseudo-solo run segment prefixes, from G_{invis} i.e., processes that their SWMR registers are not read by $q_1, ..., q_j$.

   Notice that we are replacing both p_1 and t in this construction s.t., p_1 is replaced by some process p'_1 and t is replaced some where in $\beta^*_{j,1}$ with t'. In the new run $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1) \cdot clean(j)$, that we have just generated, none of the processes $q_1, ..., q_j$ reads any SWMR register that belongs to active (not thinking) processes. Notice that $q_1, ..., q_j$ are allowed to read each other SWMR register, but still the state at the end of the run is transparent.

   Suppose processes t and t' are as described above i.e., t is read in the $j+1$'st induction step, t' is replacing t, and t was activated in some induction step $h < j+1$ after repeating the cover step (step 9) g times. Thus, $t' \in G_{invis} \subseteq G_{j,1}$, as described above, and $G_{h-1,g}$ is the set of processes to select from in the induction step h after repeating g times s.t., $t \in G_{h-1,g}$. By our construction in phase one, $G_{j,1} \subseteq G_{h-1,g}$, therefore, $t' \in G_{h-1,g}$ as well.

   In conclusion, during run $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1) \cdot clean(j)$, processes $q_1, ..., q_j$ do not read any SWMR register of active processes by the end of this run. Moreover, processes $q_1, ..., q_j$ do not read the SWMR registers of the processes in G_{invis}. Therefore, by the end of run $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1) \cdot clean(j)$, there is no "trace" of the active processes in any of the MWMR registers, and the traces in the SWMR registers are ignored. The set $G'_{invis} = G_{invis} - \{$*the replaced processes (like* t'*)*$\}$ is the set of processes to select from in the extension of the run or when replacing processes in future steps of the construction.

2. The state after some $clean(j)$ run segment in the run constructed in phase one, may be non-transparent with respect to processes that start their execution after $clean(j)$. Even though, processes $q_1, ..., q_j$ (which participate in $clean(j)$), do not read any active process in the run during $clean(j)$, the state may still be non-transparent to processes afterwards. Let a process p_2 be a process in G'_{invis}, where G'_{invis} is as defined in

case 1 above. After $clean(j)$ run segment (in run $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1) \cdot clean(j)$), p_2 starts its $p2$-segment. In this case, p_2 may read SWMR registers belong to some active processes in the run. Thus, p_2 is not running its pseudo-solo run segment after $clean(j)$.

Before constructing a run equivalent to the run of phase one, in which the "problematic" processes are not participating, we show that there are such equivalent runs. As shown in case 1 above, we construct an equivalent run by selecting processes from the currently available set of processes i.e., G'_{invis}. But, what if each process we select from G'_{invis} to participate in an equivalent run, is "discovered"? (i.e., its SWMR register is read). Therefore, we find a sub-set of G'_{invis}, none of the processes of which read the other's SWMR register, when starting its execution at $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1) \cdot clean(j)$. We construct a sub-set $G^*_{j,1}$ of the set of processes G'_{invis}, as follows: First, the processes after any transparent state perform at most k operations, when running a pseudo-solo run segment. Second, as previously shown, the length of run segment $\beta^*_{j,1}$ (as run $\beta_{j,1}$) is bounded by a function of k and d operations and it also bounds the number of processes that are writing in SWMR registers in $\beta^*_{j,1}$. We remove processes from G'_{invis} that other processes read their SWMR register s.t., there is a set of processes named $G^*_{j,1}$ of at least $\frac{|G'_{invis}|}{f(k)}$, where $f(k) = (2k+1) * (k+1)$ and $f(k)$ is constructed and calculated in [7].

In this induction step (step $j+1$), we can generate an equivalent run to $s \cdot \beta^*_{j,1} \cdot^{\sim} solo_{p'_1}(j+1)$, by selecting processes from $G^*_{j,1}$. Note that since $p'_1 \in G_{invis}$, we may need to replace it again with another process, say $p''_1 \in G^*_{j,1}$. The new run is denoted by $s \cdot \beta^{**}_{j,1} \cdot^{\sim} solo_{p''_1}(j+1) \cdot clean(j)$.

The new set of processes $G^*_{j,1} \subseteq G_{j,1}$ is used in step 7 of the first phase, instead of the set $G_{j,1} - \{p_1\}$, as written in phase one.

□

Phase 3:

Assumption B is relaxed in this phase, i.e., during WT&S operations processes are now allowed to write to different MWMR register at different orders. The proof in phases one and two fails when assumption B is relaxed in the following way. The cover we have constructed in phase one is not necessarily on the same MWMR registers that processes like p or p' are going to write to (i.e., each is allowed to write to a different set of registers).

The technique to overcome this difficulty relies on two factors, (1) the number of MWMR registers d is a constant, (2) the algorithm is a k-solo-adaptive algorithm i.e., the maximum length of all pseudo-solo run segments is k. Therefore, at a given transparent state s with respect to a large enough set of processes G, we consider the different sequences of write operations to MWMR by the different pseudo-solo run segments of processes in G. The number of different sequences of write operations to MWMR registers is bounded by a function of d and k. Each such possible sequence defines and equivalent class in G. By taking the largest equivalent class of processes we obtain a large enough subset of G such that each process in the subset performs the same sequence of writes to MWMR registers in its pseudo-solo run segment, starting at s. Such a subset is said to satisfy the *MULTI-unique* property.

Notice that at two different states s and s' that are transparent with respect to a set of processes G, any process $r \in G$ may have two different sequences of write operations to MWMR registers in its pseudo-solo runs from s and from s'. Therefore, we cannot compute in advance the required sub-set of processes. Instead, we compute the subset that corresponds to each state iteratively in rounds as described below.

We rewrite lemmas 2.5 and 2.6 without assumptions A and B as follows:

LEMMA 2.7. *Let W be a long-lived k-solo-adaptive WT&S algorithm. Then for every i, $0 \le i \le k$ and for every constant e there exists a constant $n_{e,i}$ s.t., for any set of processes G, $|G| \ge n_{e,i}$ and for any state s transparent with respect to G, there is a run segment β_i and a set of processes $G_i \subseteq G$ s.t., the following holds: (1) $|G_i| \ge e$, (2) the state at the end of run $s \cdot \beta_i$ is transparent with respect to G_i, and (3) either the first i MWMR registers written in the pseudo-solo run segments of processes in G_i, after $s \cdot \beta_i$, are the same and are covered in $s \cdot \beta_i$, or, the pseudo solo runs starting at $s \cdot \beta_i$ have less than i writes to the same MWMR registers and all these writes are covered in $s \cdot \beta_i$. And in addition, there is a large set of runs* equivalent *to β_i s.t., each run β'_i equivalent to β_i is with respect to a large enough set $G'_i \subseteq G_i$.*

Sketch of proof of the inductive step in Lemma 2.7:

If a large enough subset of processes write to less than $j+1$ MWMR registers in their pseudo-solo run segments, then we show that there is a large enough subset of processes which are writing to the same j MWMR registers in the same order, and this set has a cover. Otherwise we show that there is a set of $j+1$ MWMR registers for which there is a large enough set of processes that write on them in the same order in their pseudo-solo run segments and this set has a cover, in state s.

All steps of the proof, but steps 3, 4 and 5 are the same as in phase one and two. Steps 4 and 5 are the same as in phase one, but here instead of referring to assumption B, we argue by using the MULTI-unique property, as explained below. Unless stated otherwise we use the same notations and symbols as in phases one and two.

MULTI-unique:

In step 3 of phase one, before selecting process p_1, we should first reduce the set $G_{j,1}$ to a sub-set $\hat{G}_{j,1}$. The set $\hat{G}_{j,1}$ is selected s.t., every process in this set is writing in the same MWMR registers and at the same order, when executing its pseudo-solo run segment, starting after $s \cdot \beta_{j,1}$.

The detailed construction is as follows: In each iteration of steps 3- 7 in each inductive step the corresponding set $\hat{G}_{j,1}$ (which was $G_{j,1}$ in previous phases) is constructed in several rounds. In each round we select a sub-set of the subset selected in the previous round. Let Q_0 stand for $G_{j,1}$ and let the set of processes after round h be $Q_h \subseteq Q_0$. In round $h+1$, we observe the $h+1$'st write operation of all pseudo-solo run segments of processes in Q_h (the previous round), starting after $s \cdot \beta_{j,1}$ and we construct the new set Q_{h+1}, as follows:

1. If more than half of the processes do not have a pseudo-solo run segment with $h+1$ write operations in MWMR registers before eating, then we define Q_{h+1} to be these processes (i.e., $|Q_{h+1}| \geq \frac{1}{2}|Q_h|$) and Q_{h+1} is $\hat{G}_{j,1}$, the set of processes that holds the Multi-unique property at $s \cdot \beta_{j,1}$.
2. Else, we denote the rest of the processes by Q'_{h+1} and $|Q'_{h+1}| \geq \frac{1}{2}|Q_h|$. Thus, processes in Q'_{h+1} have an $h+1$'st write operation in MWMR registers. Therefore, there is at least one MWMR register that a set of size $\frac{|Q'_{h+1}|}{d}$ is writing in it. The recursive function is $|Q_{h+1}| \geq \frac{|Q'_{h+1}|}{d} \geq \frac{|Q_h|}{2d}$.

Since there are at most k rounds, then $|Q_k| \geq \frac{|Q_0|}{[2d]^k}$. Finally, we denote $\hat{G}_{j,1} = Q_k$ thus, $|\hat{G}_{j,1}| \geq \frac{|G_{j,1}|}{[2d]^k}$, (since $Q_0 = G_{j,1}$).

□

3. CONCLUDING REMARKS

In the run we constructed in the main theorem's proof, each process participates at most once. I.e., No process in the constructed run invokes a WT&SET operation twice. One might think that therefore the proof is for single-shot WT&S objects, but no. In the constructed run processes that attempt to capture the WT&S bit and fail are assumed to leave the WT&S object in an idle state. That is, the object may be captured and released several times, namely is long-lived. Processes, however, access it each at most one time. Perhaps a more sophisticated construction would reuse the processes and prove a stronger bound for long-lived objects in which the same set of processes access the object multiple times.

In a single shot WT&S object, the object may be captured at most one time. Furthermore, if some processes attempted to capture the WT&Set bit and failed, then no later process will succeed in capturing the bit. Notice that such an object has an adaptive implementation with just two MWMR registers following the X and Y MWMR registers construction used by Lamport in [10] (and which is the basis of the splitter object [11]).

There is an implementation of adaptive collect with a function of n MWMR registers, but the most space efficient adaptive renaming object has an implementation with $N >> n$ MWMR registers, where N is the total number of processes in the system, while n is the maximum number of concurrent processes. This leaves an open question, if there is an adaptive WT&S algorithm for implementing renaming object in the Read/Write Shared Memory model that uses a function of n MWMR registers?

Acknowledgments:
We are in debt to Gideon Stupp for essential discussions and collaboration during the early phases of this research. We would like also to thank Hagit Attiya, Arie Fouren, Eli Gafni, Michael Merritt and Gadi Taubenfeld for many helpful discussions.

4. REFERENCES

[1] James E. Burns and Nancy A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, December 1993.

[2] H. Attiya and A. Fouren. Adaptive long-lived renaming with read and write operations. Technical Report 0956, Faculty of Computer Science, Technion, Haifa, 1999. http://www.cs.technion.ac.il/~hagit/pubs/tr0956.ps.gz.

[3] Y. Afek, G. Stupp, and D. Touitou. Long lived adaptive collect with applications. In *Proc. of the 40th IEEE Ann. Symp. on Foundation of Computer Science*, pages 262–272, October 1999.

[4] Y. Afek, H. Attiya, G. Stupp, and D. Touitou. Adaptive long-lived renaming using bounded memory. Submitted to DISC99. ftp://ftp.math.tau.ac.il/pub/stupp/PAPERS/name99.ps.gz, 1999.

[5] James H. Anderson and Jae-Heon Yang. Time/contention trade-offs for multiprocessor synchronization. *Information and Computation*, 124(1):68–84, 10 January 1996. Also in STOC94, 224-233.

[6] Y. Afek, E. Gafni, and M. Merritt. Adaptive algorithms utilyzing adaptive collect and snapshot. Submitted for publication, ftp://ftp.math.tau.ac.il/pub/afek/1shotRn.ps, 1999.

[7] Y. Afek, P. Boxer, and D. Touitou. Bounds on the shared memory requirements for long-lived & adaptive objects. Draft of full paper. ftp://ftp.math.tau.ac.il/pub/afek/abt.ps.

[8] H. Attiya and A. Fouren. Adaptive long-lived renaming with read and write operations. Extended Abstract, November 1998.

[9] Y. Afek, H. Attiya, A. Fouren, G. Stupp, and D. Touitou. Long-lived renaming made adaptive. In *Proc. 18th Annual ACM Symp. on Principles of Distributed Computing*, pages 91–103, May 1999.

[10] L. Lamport. A fast mutual exclusion algorithms. *ACM Trans. on Computer Systems*, 5(1):1–11, February 1987.

[11] M. Moir and J. Anderson. Wait-free algorithms for fast, long-lived renaming. *Science of Computer Programming*, 25(1):1–39, October 1995. Also in Proc. 8th Int. Workshop on Distributed Algorithms, September 1994, 141-155.