Simultaneous Consensus Tasks: A Tighter Characterization of Set-Consensus

Yehuda Afek[1], Eli Gafni[2], Sergio Rajsbaum[3], Michel Raynal[4], and Corentin Travers[4]

[1] Computer Science Department, Tel-Aviv University, Israel 69978.
afek@math.tau.ac.il
[2] Department of Computer Science, UCLA, Los Angeles, CA 90095, USA.
eli@cs.ucla.edu
[3] Instituto de Matemáticas, UNAM, D. F. 04510, Mexico.
rajsbaum@math.unam.mx
[4] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France.
raynal|ctravers@irisa.fr

Abstract. We address the problem of solving a task $T = (T_1, ...T_m)$ (called $(m, 1)$-BG), in which a processor returns in an arbitrary one of m simultaneous consensus subtasks $T_1, ...T_m$. Processor p_i submits to T an input vector of proposals $(prop_{i,1}, ..., prop_{i,m})$, one entry per subtask, and outputs, from just one subtask ℓ, a pair $(\ell,\ prop_{j,l})$ for some j. All processors that output at ℓ output the same proposal.

Let d be a bound on the number of distinct input vectors that may be submitted to T. For example, $d = 3$ if Democrats always vote Democrats across the board, and similarly for Republicans and Libertarians. A wait-free algorithm that immaterial of the number of processors solves T provided $m \geq d$ is presented. In addition, if in each T_j we allow k-set consensus rather than consensus, i.e., for each ℓ, the outputs satisfy $|\{j \mid prop_{j,\ell}\}| \leq k$, then the same algorithm solves T if $m \geq \lceil d/k \rceil$.

What is the power of $T = (T_1, ..., T_m)$ when given as a subroutine, to be used by any number of processors with any number of input vectors? Obviously, T solves m-set consensus since each processor p_i can submit the vector $(id_i, id_i, ...id_i)$, but can m-set consensus solve T? We show it does, and thus simultaneous consensus is a new characterization of set-consensus.

Finally, what if each T_j is just a binary-consensus rather than consensus? Then we get the novel problem that was recently introduced of the Committee-Decision. It was shown that for 3 processors and $m = 2$, the simultaneous binary-consensus is equivalent to $(3, 2)$-set consensus. Here, using a variation of our wait-free algorithms mentioned above, we show that a task, in which a processor is required to return in one of m simultaneous binary-consensus subtasks, when used by n processors, is equivalent to (n, m)-set consensus. Thus, while set-consensus unlike consensus, has no binary version, now that we characterize m-set consensus through simultaneous consensus, the notion of binary-set-consensus is well defined. We have then showed that binary-set-consensus is equivalent to set consensus as it was with consensus.

1 Introduction

The Borowsky-Gafni simulation scheme relies on the realization that there is a read-write algorithm by which n processors involved in n simultaneous sub-consensus-tasks $T_1, ..., T_n$, can reach consensus in a wait-free manner in at least some T_k, though k is unknown a priori. Thus we can define the $(m, 1)$-BG task: processor p_i starts with some input value v_i and has to output a pair (ℓ, v_j) for some $1 \leq \ell \leq m$, and v_j is the initial proposal of some p_j in the participating set. All processors that output with first argument ℓ have to output the same value.

One can think of a variation of $(m, 1)$-BG in which the inputs are m-vectors and processors that output at T_k are to output the same k entry from one of the vectors. But it is easy to see that the vector problem solves the value problem by each processor p_i inputing $(v_i, v_i, ..., v_i)$, as well as the value problem solving the vector problem by associating vectors with values, and then for value v_i when output at k, a processor substitutes the kth entry of the associated vector. Henceforth the presentation proceeds with the value version.

In the BG simulation [3, 4], we use n agreement protocols and rely on the fact that if the first agreement is not resolved then there is a processor "stuck" in the middle of the first agreement protocol and consequently we can proceed with one processor less. Here, when we have n proposals rather than n processors, we show a variant agreement protocol by which in each agreement protocol that does not terminate we, lose a proposal rather than a processor. Thus a sequence of n agreement protocols will solve the $(m, 1)$-BG task, $m \geq n$.

Now that we generalized the $(m, 1)$-BG task to any number of processors, we investigate the relationship between m and the power of the consensus that each task provides. Suppose that in each task T_j, we do not require consensus but rather k-set consensus. Thus, we have m subtasks $T_1, ..., T_m$ and processors output (ℓ, v_j) for some $1 \leq \ell \leq m$ and for each $\ell : |\{v_j \,|(\ell, v_j) \in \mathit{output}\,\}| \leq k$. We call this task (m, k)-BG.

Our second result is that (m, k)-BG is read-write wait-free solvable for any number of processors, if the number of initial choices d satisfies $m \geq \lceil d/k \rceil$. Thus if we allow each T_j to solve 2-set consensus, then m can be half the number of initial choices. Alternatively, it can just be reduced to the consensus case: just solve $(m, 1)$-BG and group the outputs 1 to k, $k + 1$ to $2k$, etc.

Until this point we investigated what variation of BG tasks can be solved wait-free. We then turn to BG tasks with parameters that do not render it solvable and wonder about the power of these tasks.

Suppose we are given an $(m, 1)$-BG task as a subroutine. Since each subtask does consensus, it trivially solves m-set consensus by ignoring the subtask index. Can m-set consensus solve m-BG? Notice that $(m, 1)$-BG associates different output values with different subtasks. Our $(m, 1)$-BG algorithm answers this question on the affirmative. By using m-set consensus, the number of initial choices n becomes m, and then we can wait-free solve the $(m, 1)$-BG.

What if each subtask in the $(m, 1)$-BG task is a binary-consensus rather than consensus? We refer to this problem as m-BG-Binary. If $m = 1$ then we have our beloved consensus and it is known how to transform binary-consensus into consensus by repeated consensus on the binary representation of the eventual output value (A different approach is presented in [14].). But what if $m = 2$? When we try repeated binary-consensus, at the first invocation p_i may get a value from T_1 and in the second from T_2. How do you build a prefix under these conditions?

The question of the $(m, 1)$-BG task when each subtask is a binary consensus and the input is a binary vector with entry for each subtask was recently investigated in [10, 11]. Thus, in subtask T_j if all input values to T_j are 0, only 0 can be returned for T_j. The problem was called the m-Committee-Decision problem as the connection to $(m, 1)$-BG was not realized. Obviously BG tasks encompass Committee-Decision as the proposed values are vectors and when returning a vector for T_j, one projects on the jth entry. Thus the interesting direction is to show that Committee-Decision encompasses BG tasks.

Using explicit topological arguments, it was shown in [10, 11] that 2-Committee-Decision when used by 3 processors is equivalent to $(3, 2)$-set consensus. Here, as a simple corollary we show that $(m, 1)$-BG for n processors is equivalent to (n, m)-set consensus. Thus we show the equivalence between BG tasks and Committee-Decision.

The paper is organized as follows. We first outline the various tasks we deal with (section 2). We then outline the rather simple agreement algorithm that wait-free solves $(m, 1)$-BG for $m \geq n$ (sections 3 and 4). We then show a bit more involved construction that reduced $(m, 1)$-BG to m-Committee-Decision, or alternatively referred to as m-BG-Binary (section 5). We conclude with a discussion of the merits of characterizing set-consensus through simultaneous-consensus (section 6).

2 Problems Definitions and Preliminaries

In all the paper, we are interested in wait free algorithms [12].

2.1 Computational Model

Processor model The system consists of an arbitrary number of processors [9, 15] that we denote $p_1, p_2, \ldots$ In a run a *participating* processor p_i wakes up with some initial value $input_i$. The inputs value are taken from a set $Input$ of size n. It is important to notice that n denotes the maximal number of values participating processors wake up with. The number of processors that participate in a run is unknown to the processors.

A processor can crash. Given a run, a processor that crashes is said to be *faulty*, otherwise it is *correct* in that execution. Each processor progresses at its own speed, which means that the system is asynchronous.

Coordination model The processors communicate and cooperate through atomic multi-reader/multi-writer registers. To simplify algorithm descriptions, *write-snapshot* objects [1, 3] are also available to the processors.

A write-snapshot WS object provides the processors with a single operation denoted WRITESNAPSHOT(). It is a one-shot object in the sense that each processor can invoke WS at most once. A processor p_i invokes WS.WRITESNAPSHOT(v_i), and if it does not crash during the invocation, obtains a set of value s_i. The sets returned satisfy the two following properties:

– Self containment: $v_i \in s_i$,
– Comparability : $\forall i, j : i \neq j \Rightarrow s_i \subseteq s_j \vee s_j \subseteq s_i$.

Such an object can be implemented on top of multiple-reader/multiple-writer registers for an arbitrary number of processors [7].

2.2 The Problems

(m,1)-BG In the $(m, 1)$-BG problem, processors are trying to simultaneously solve m instances of the consensus problem. Each processor is required to decide in at least one of these instances. There are m consensus subtasks $T_1, ..., T_m$. Processor p_i wakes up with a private value v_i and is required to return a pair (ℓ, v_j) such that $1 \leq \ell \leq m$ and the value v_j has been proposed by some p_j. All processors that return first argument ℓ have to agree and return the same v_j. More precisely, each processor has to decide a pair (ℓ, v) such that:

– Termination: No processor takes infinitely many steps without deciding.
– Validity: If a processor p_i decides (ℓ, v_j) then $\exists j$ such that processor p_j wakes up with value v_j.
– Agreement: $\forall \ell, 1 \leq \ell \leq m : |\{v_j : (\ell, v_j)$ is decided by some processor $\}| \leq 1$.

(m,k)-BG The (m, k)-BG task is a generalization of the $(m, 1)$-BG problem. As in $(m, 1)$-BG, processors have to return a pair (ℓ, v). The processors that return first argument ℓ may return cumulatively at most k distinct values. The pairs returned have to satisfy the validity and termination properties of the $(m, 1)$-BG problem and the following agreement property:

– $\forall \ell, 1 \leq \ell \leq m : |\{v_j : (\ell, v_j)$ is decided by some processor $\}| \leq k$.

k-Set Consensus The k-set consensus problem is a generalization of consensus where processors must decide on at most k different values that have been previously proposed [5]. When $k = 1$, the problem boils down to the standard consensus problem [6]. Each processor is required to decide a value subject to the following conditions:

– Agreement: at most k distinct values are decided.
– Termination: no processor takes infinitely many steps without deciding.
– Validity: a decided value is an initial input value for some participating processor.

It is shown in [2, 13, 16] that in a system of $\alpha > k$ processors, the k-set consensus problem has no wait free solution when processors may have distinct input values.

m-Committee-Decision or m-BG-Binary In the binary consensus problem, processors start with either 0 or 1 and are required to eventually agree on one of their initial value. Suppose now that processors are provided with a collection of binary consensus objects $B_1, \ldots, B_m$ but are not guaranteed to obtain a response from each object, even if they propose a value in each binary consensus. A processor p_i is only guaranteed to obtain a response from one object B_j and j is not known a priori. Moreover, j may change from invocation to invocation.

More precisely, this coordination scheme is captured by the m-Committee-Decision problem [11]. In the m-Committee-Decision problem, processors are trying to solve m binary consensus instances called committees and each processor is required to make a decision for at least one of them. More explicitly, each processor p_i initially proposes a vector $V_i \in \{0,1\}^k$ (i.e., $V_i[c], 1 \leq c \leq k$ is p_i's proposal for the c-th committee) and decides a pair (c, v) such that:

- Termination: No processor takes infinitely many steps without deciding.
- Validity: If a processor decides (c, v) then $\exists j$ such that $v = V_j[c]$.
- Agreement: Let p_i and p_j be two processors that decide (c_i, v_i) and (c_j, v_j) respectively. $c_i = c_j \Rightarrow v_i = v_j$.

3 Wait-Free Solution to $(m, 1)$-BG, n Initial Values, $m \geq n$

Processor p_i marches in order through T_1 followed by T_2, etc. In T_i a processor writes an input value to its cell. The input to T_1 is the input it wakes up with. The input to T_j is adopted from T_{j-1}.

At T_j a processor writes its input, returns an atomic snapshot of input values and posts its snapshot in shared memory. If it then sees a snapshot of values of cardinality one, it returns this value for T_j and quits. Else, it adopts the minimum value from one of the posted snapshots (maybe its own) and proceeds with it to T_{j+1} (figure 1).

The observation is that the number of distinct values proposed to T_j is at most $n - (j - 1)$, thus a processor that arrives at T_n is guaranteed to get a snapshot of size one at T_n and to return.

in shared memory: $WS[1, \ldots, m]$; array of write-snapshot objects.
$SS[1, \ldots, m][1, \ldots, m]$ array of mwmr registers, initially $\perp$.

function $(m, 1)$-BG(v_i)
$est_i \leftarrow v_i$;
for $r_i = 1$ **to** m **do**
  $S_i \leftarrow WS[r_i]$.WRITESNAPSHOT(est_i);
  $SS[r_i, |S_i|] \leftarrow S_i$;
  for $\ell = 1$ **to** m **do** $ss[\ell] \leftarrow SS[r_i, \ell]$ **enddo**;
  if $ss[1] \neq \perp$ **then** return$(r_i, ss[1])$
    else $est_i \leftarrow \min(ss[\ell])$ s.t. $(\ell \in \{1, \ldots, m\}) \wedge (ss[\ell] \neq \perp)$
  endif
enddo

Fig. 1. $(m, 1)$-BG algorithm, n initial value, $m \geq n$, code for p_i.

4 Wait-Free Solution to (m, k)-BG, n Initial Values, $m \geq \lceil \frac{n}{k} \rceil$

At each T_i, a processor tries to choose a value that appears in a snapshot of size k or less. The observation is that going from T_j to T_{j+1} at least k values are left behind. The algorithm is described in figure 2.

in shared memory: $WS[1, \ldots, m]$; array of write-snapshot objects.
$SS[1, \ldots, m][1, \ldots, m]$ array of mwmr registers, initially $\perp$.

function (m, k)-BG(v_i)
(01) $est_i \leftarrow v_i$;
(02) **for** $r_i = 1$ **to** m **do**
(03) $S_i \leftarrow WS[r_i]$.WRITESNAPSHOT(est_i);
(04) $SS[r_i, |S_i|] \leftarrow S_i$;
(05) **for** $\ell = 1$ **to** m **do** $ss[\ell] \leftarrow SS[r_i, \ell]$ **enddo**;
(06) **if** $\exists \ell, 1 \leq \ell \leq k : ss[\ell] \neq \perp$ **then** return$\big(r_i, \min(ss[\ell])\big)$
(07) **else** $est_i \leftarrow \min(ss[\ell])$ s.t. $(\ell \in \{1, \ldots, m\}) \wedge (ss[\ell] \neq \perp)$
(08) **endif**
(09) **enddo**

Fig. 2. (m, k)-BG algorithm, n initial values, $m \geq \lceil \frac{n}{k} \rceil$, code for p_i.

4.1 Proof of the Protocol

In the following, we say that a value v is proposed at stage $r, 1 \leq r \leq m$ if it exists a processor p_i that starts stage r with $est_i = v$. For each r, $1 \leq r \leq m$, let $I[r]$ be the set of values proposed at stage r.

Lemma 1. *(Validity) Let (ℓ, v) be a pair decided by some processor. v is a proposed value.*

Proof Let p_i be a processor that decides (ℓ, v) at stage r. Let us observe v is taken from the set of input values of stage r, i.e., $v \in I[r]$. Moreover, $\forall r', 2 \leq r' \leq m$, $I[r'] \subseteq I[r' - 1]$ (line 07). As $I[1] =$ the set of values the processors wake up with and $v \in I[r] \subseteq I[1]$, validity follows. $\Box_{Lemma\ 1}$

Lemma 2. *(Termination) A correct processor eventually decides.*

Proof We first observe that $\forall r, 1 \leq r \leq m : |I[r]| \leq n - k(r-1)$ (Observation *O1*). Let us assume for contradiction that there is a correct processor p_i that does not decide. This means that p_i marches through stages $1, 2, \ldots, m$ without deciding. In particular, at stage m, p_i obtains a snapshot $S_i \subseteq I[m]$. It follows from *O1* that $|S_i| \leq |I[m]| \leq n - k(m-1)$. Moreover, as $m \geq \lceil n/k \rceil$, we obtain

$|S_i| \leq n - k(\lceil n/k \rceil - 1) \leq k$, from which we conclude that p_i decides at stage m (line 06): a contradiction.

Observation O1 $\forall r, 1 \leq r \leq m : |I[r]| \leq n - k(r-1)$.
Proof of O1 As there are at most n proposed values and these values are the input ones at stage 1, $|I[1]| \leq n$. Let us assume that the observation is true at stage $r, 1 \leq r < m$. Let p_i be a processor that proposes a value at stage $r+1$. At stage r, p_i updates its estimate with a value picked in a snapshot of size $> k$. Moreover, there are at most $|I[r]| - k$ such snapshots and for each of them, only one value can be picked by the processors (line 07). Consequently, at most $|I[r]| - k$ values can be proposed at stage $r+1$, from which we obtain $|I[r+1]| \leq |I[r]| - k \leq n - kr$. *End of the proof of O1*

$\Box_{Lemma\ 2}$

Lemma 3. *(Agreement)* $\forall r, 1 \leq r \leq m : |\{v : \exists p_i$ *that decides* $(r, v)\}| \leq k$.

Proof Let r be a stage number. The values decided by processors that return at stage r are picked in a snapshot of size k or less (line 06). Since these snapshots contain cumulatively at most k distinct values, at most k distinct values are decided at stage r.

$\Box_{Lemma\ 3}$

5 $(m,1)$-BG from m-BG-Binary

Let the number of initial values be $n > m$. We show how to use $(n-1)$-BG-Binary to reduce the number of initial values by at least 1 to $n-1$. Obviously m-BG-Binary implements j-BG-Binary for all $j \geq m$.

Thus the scheme is to start with the n initial values, reduce it to $n-1$ then to $n-2$ and until m. At this point we have at most m initial values and we can wait free solve $(m,1)$-BG.

To reduce the number of initial values from n to $n-1$, we go through $n-1$ stages $T_1, ..., T_{n-1}$. In each stage we post initial value, snapshot, post snapshot, and then read snapshots. The algorithm is described in figure 3.

If a processor sees posted snapshot of size 1 containing some v_j but no snapshot of size 2, then it returns v_j. Otherwise it adopts the smallest value in some snapshot of size 2 or more and continues to the next stage.

If a processor finishes stage T_{n-1} without returning, it invokes the $(n-1)$-BG-Binary object. The observation to make is that in all stages there are posted snapshots of size 2. Otherwise 2 values would have been left behind at some stage and the processor should have terminated by the end of stage T_{n-1}.

Now come the voting step in which the processor goes to the $n-1$-BG-Binary object. At committee j it will observe the snapshot posted at T_j. There is a snapshot of size 2 containing two values. We associate the smaller value with 0 and the larger with 1. If the processor also sees a snapshot of size 1 posted, it votes for that value. Thus a processor that quits without voting is guaranteed that the value it choses for T_j will be voted for by all.

in shared memory: $WS[1,\ldots,m]$ array of write-snapshot objects
$SS[1,\ldots,m][1,\ldots,m+1]$ array of mwmr registers, initially $\perp$

function $(m,1)$-BGFROMBGBINARY(v_i)
(01) $est_i \leftarrow v_i$;
(02) **for** $r_i = 1$ **to** m **do**
(03) $S_i \leftarrow WS[r_i]$.WRITESNAPSHOT(est_i);
(04) $SS_i[r_i, |S_i|] \leftarrow S_i$;
(05) **for** $j = 1$ **to** m **do** $ss[j] \leftarrow SS[r_i, j]$ **enddo**;
(06) **if** $(ss[1] \neq \perp) \wedge (ss[2] = \perp)$ **then** return$(r_i, ss[1])$
(07) **else** $est_i \leftarrow \min(ss[j])$ s.t. $(j \in \{2,\ldots,m\}) \wedge (ss[j] \neq \perp)$
(08) **endif**
(09) **enddo**
% If p_i has not succeeded in $T_1,\ldots,T_m$, it uses m-BG Binary to decide %
(10) **foreach** $r \in \{1,\ldots,m\}$ **do**
(11) **let** v_m (resp. v_M) be the smallest value (resp. greatest) value in $SS[r,2]$;
(12) **case** $(v_m \in SS[r,1])$ **then** $V_i[r] \leftarrow 0$
(13) $(v_M \in SS[r,1])$ **then** $V_i[r] \leftarrow 1$
(14) *default* **then** $V_i[r] \leftarrow 0$ or 1 arbitrarily
(15) **endcase**
(16) **enddo**
(17) $(c_i, d_i) \leftarrow m$-BGBINARY(V_i);
(18) **if** $d_i = 1$ **then** return$\big(c_i, \max(SS[c_i,2])\big)$ **else** return$\big(c_i, \min(SS[c_i,2])\big)$ **endif**

Fig. 3. $(m,1)$-BG from m-BG-Binary, n initial values, $n = m+1$, code for p_i.

5.1 Proof of the Protocol

We first prove the observation stated in the algorithm description (Lemma 4). Wait-free termination directly follows from the protocol text. We use Lemma 4 in the proofs of validity (Lemma 5) and agreement (Lemma 6).

Lemma 4. *Let p_i be a processor that returns at line 18. When p_i reads $SS[1,2]$, $SS[2,2]$, …, $SS[m,2]$ at line 11, we have $\forall 1 \leq r \leq m : SS[r,2] \neq \perp$.*

Proof Let us assume for contradiction that the lemma is false. This means that it exists a process p_i that returns at line 18 and a stage number $R, 1 \leq R \leq m$ such that p_i does not see a snapshot of size 2 posted at stage R. More precisely, when p_i reads $SS[R,2]$ in the second phase of the protocol (line 11), $SS[R,2] = \perp$. Let τ be the time at which this occurs. As a processor can post in $SS[R,2]$ only a snapshot of size 2 obtained at stage R (line 04), it follows that $\forall \tau' \leq \tau : SS[R,2] = \perp$.

As p_i proceeds to the second phase of the protocol, it tries to decide in each T_r, $1 \leq r \leq m$. We show that that p_i decides in the first phase of the protocol (at line 06): a contradiction. The proof consider two cases according to the value of R.

– $m = R$. Let us observe that the first phase of the protocol is the (m,k)-BG protocol instantiated with $k = 1$ in which processors wake up with at most $n = m+1$ values.

Consequently, observation *O1* stated and proved in Lemma 2 is still valid. It then follows that at most $(m+1)-(m-1)=2$ values can be proposed at stage m.

As p_i proceeds to the second phase of the algorithm, it obtains a snapshot at stage m. Moreover, when p_i tries to decide at stage r, $SS[r,2]=\perp$. Consequently, p_i obtains a snapshot of size 1 and does not see a snapshot of size 2, from which we conclude that p_i decides at line 06 in the first phase of the algorithm.

– $m > R$. Let us first remark that at most $m-R$ values can be proposed at stage $R+1$ before time τ. The values proposed at stage $R+1$ are taken among the smallest values in snapshots of size ≥ 2 posted at stage R. As at most $(m+1)-(R-1)$ values are proposed at stage R (Observation *O1* in Lemma 2), at most $(m+1)-(R-1)$ distinct snapshots can be posted in that stage. Moreover, as values proposed at stage $R+1$ are picked in snapshots of size > 1 and no snapshot of size 2 is posted before time τ ($SS[R,2]=\perp$ before time τ), it follows that at most $(m+1)-(R-1)-2=m-R$ values can be proposed in stage $R+1$ before time τ.

  We can think of stages $T_{R+1},\dots,T_m$ as a $(m-R,1)$-BG protocol. It follows from the remark above that, before time τ, the size of the set of input values to this $(m-R,1)$-BG protocol is at most $m-R$. As this protocol solves the $(m-R,1)$-BG task if the number of distinct input values is $\leq m-R$ (section 3), a processor cannot marches through $T_{R+1},\dots,T_m$ before time τ without deciding. Hence, as p_i tries to decide in $T_{R+1},\dots,T_m$ before time τ, p_i decides in some T_r at line 06.

$\square_{Lemma\ 4}$

Lemma 5. *(Validity) Let (ℓ,v) be a pair decided by some processor. v is a proposed value.*

Proof Let p_i be a processor that decides (ℓ,v). If p_i decides in the first phase of the protocol (at line 06), v is contained in a posted snapshot of size 1. If p_i decides in the second part of the protocol, it follows from line 18 and Lemma 4 that v is contained in a posted snapshot of size 2. In both cases, v belongs to some snapshot posted in the first phase of the protocol.

As already observed, the first part of the protocol is the $(m,1)$-BG protocol. As the proof of validity in the $(m,1)$-BG protocol does not depend on the number of values processors wake up with (Lemma 1), we can reuse it here. In particular, it is shown in Lemma 1 that all posted snapshots are included in the set of values processors wake up with, from which we conclude that v is a proposed value. $\square_{Lemma\ 5}$

Lemma 6. *(Agreement) $\forall \ell, 1\leq \ell \leq m$: p_i returns (ℓ,v_i) and p_j returns (ℓ,v_j) $\Rightarrow v_i=v_j$.*

Proof In the following, we say that a processor p_i decides in slot ℓ if it returns (ℓ,v) at line 06 or at line 18. We show that for any slot $\ell, 1\leq \ell\leq m$, at most one value is decided. Let D_ℓ be the set of processes that decide in slot ℓ. Let us consider a slot ℓ such that $D_\ell \neq \emptyset$. We consider three cases:

– Each processor p_i that belongs to D_ℓ returns at line 06. Due to the atomic snapshot properties, at most one snapshot that contains only one value can be returned by the object $WS[\ell]$. It then follows from lines 06-07 that processors $\in D_\ell$ decide the same value.
– Each processor that belong to D_ℓ returns at line 18. This means that each processor $p_i \in D_\ell$ gets back a pair (ℓ, d_i) from the m-BGBINARY object. Due to the agreement property of the object, $\exists d \in \{0, 1\}$ such that $\forall p_i \in D_\ell, d_i = d$.
  Moreover, due to Lemma 4, when $p_i \in D_\ell$ reads $SS[\ell, 2]$ at lines 11 and 18, $SS[r, \ell] \neq \perp$. It then follows from line 18 and the fact that $\exists d$ such that $\forall p_i \in D_\ell :$ $d_i = d$ that each processor that belongs to the set D_ℓ chooses the same value in $SS[\ell, 2]$ and agreement follows.
– Some processors that belong to D_ℓ return at line 06 and other processors at line 18. Let C be the set of processors that invoke the m-BGBINARY object (a processor in C does not necessarily decides in slot ℓ). Among them, let p_c be the first processor that reads $SS[\ell, 1]$. This occurs at time τ. If p_c sees a value v, every processor in C proposes v for committee ℓ (lines 12-13). Therefore, v is the only value that can be decided in slot ℓ through the m-BGBINARY object and agreement follows.
  Suppose that p_c does not see a snapshot of size 1 ($SS[\ell, 1] = \perp$) in slot ℓ. We claim that no process can decide at line 06 in slot ℓ: a contradiction with the case assumption. To prove the claim, let us observe that when p_c reads $SS[\ell, 1]$ (lines 12-12), $SS[\ell, 2] \neq \perp$ (Lemma 4). Thus, a process that subsequently reads $SS[\ell, 1] \neq \perp$ reads also $SS[\ell, 2] \neq \perp$ and cannot decide in slot ℓ at line 06.

$\Box_{Lemma\ 6}$

6 Conclusion

Simultaneous consensus was first introduced in [10, 11] where it was shown using explicit topological arguments that 3 processors two committees is equivalent to $(3, 2)$-set consensus. The approach of interpreting algorithms through the prism of simultaneous consensus was then followed in [8] where it proved beneficial in obtaining a clear proof of robustness. Here, we close the circle. We utilize the observation that the BG simulation [2, 4] is also about simultaneous consensus, to adopt a completely algorithmic approach to the question. Through this algorithmic approach that adopts ideas from BGs, we show that simultaneous consensus in a clear way captures consensus and set consensus. Moreover, it is a stronger paradigm than set-consensus. It trivially implements set-consensus, but it took some work to show that set consensus implements it. We expect this new view of set-consensus to prove beneficial in the future.

Atomic-Snapshots Shared-Memory is a higher level construct than SWMR Shared-Memory, and yet equivalent to it. Later Immediate-Snapshot Memories were proved to be even a higher level construct than Atomic-Snapshots. There, when "higher level" can be interpreted precisely as "less executions" it is a consequence of [2, 13, 16] that Immediate-Snapshots is the end of the road. Is simultaneous consensus the end of the road for set-consensus? Will there be a sense in which one may find even a tightest characterization of set-consensus? While we leave this question open, we feel that at the least it is now easier to motivate set-consensus through simultaneous consensus. Simultaneous consensus comes across as a bit less of “an invention of bored theorists,” than the question of “electing multiple values.” Multiple fronts is natural in life while multiple-leaders is less so.

References

1. Afek Y., H. Attiya, Dolev D., Gafni E., Merrit M. and Shavit N., Atomic Snapshots of Shared Memory. *Proc. 9th ACM Symposium on Principles of Distributed Computing (PODC’90)*, ACM Press, pp. 1–13, 1990.
2. Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations *Proc. 25th ACM Symposium on the Theory of Computing (STOC’93)*, ACM Press, pp. 91-100, 1993.
3. Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming (Extended Abstract). *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC’93)*, ACM Press, pp. 41-51, 1993.
4. Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG Distributed Simulation Algorithm. *Distributed Computing,* 14(3):127–146, 2001.
5. Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.
6. Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.
7. Gafni E., Group-Solvability. *Proc. 18th Int. Symposium on Distributed Computing (DISC’04)*, Springer Verlag LNCS #3274, pp. 30–40, 2004.
8. Gafni E. and Kouznetsov P., Two Front Agreement with Application to Emulation and Robustness. *to appear*.
9. Gafni E., Merritt M. and Taubenfeld G., The Concurrency Hierarchy, and Algorithms for Unbounded Concurrency. *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC’01)*, ACM Press, pp. 161–169, 2001.
10. Gafni E. and Rajsbaum S., Musical Benches. *Proc. 19th Int. Symposium on Distributed Computing (DISC’05)*, Springer Verlag LNCS # 3724, pp. 63–77, September 2005.
11. Gafni E. and Rajsbaum S., Raynal M., Travers C., The Committee Decision Problem. *Proc. Theoretical Informatics, 7th Latin American Symposium (LATIN’06)*, Springer Verlag LNCS #3887, pp. 502-514, 2006.
12. Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on programming Languages and Systems*, 11(1):124-149, 1991.
13. Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.
14. Mostefaoui A., Raynal M.and Tronel F., From Binary Consensus to Multivalued Consensus in Asynchronous Message-Passing Systems. *Information Processing Letters*, 73:207-213, 2000.
15. Merrit M., Taubenfeld G., Computing with infinitely many processes. *Proc. 14th Int. Symposium on Distributed Computing (DISC’00)*, Springer Verlag LNCS #1914, pp. 164–178, October 2000.
16. Saks, M. and Zaharoglou, F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.