


[Sch96] Eric Schenk. The consensus hierarchy is not robust, manuscript, 1996.


6 Conclusion and Further Research

In this paper, we have studied the robustness of the $h_m^r$ hierarchy for the class of hard-wired objects under the $M_1$ connection scheme. We showed that this hierarchy is not robust, by introducing a deterministic hard-wired conditional consensus object, denoted as $W$ (whose consensus number is at most 3), which demonstrates that objects at level 3 of $h_m^r$ can implement consensus for any number $N$ of processes. The current knowledge as for the robustness of the various classes of $h_m^r$ is as follows:

1. $h_m^r$ is not robust for the class of hard-wired objects (either deterministic or non-deterministic) using $M_1$ connection scheme ([CHJT94a] and the current work).

2. The question for non-deterministic soft-wired types and for non-deterministic hard-wired using $M_2$ or $M_3$ connection rules is still open.

A natural question is whether a simple modification of our technique can be used to prove that $h_m^r$ is not robust for $M_2$ or $M_3$ objects. The following lemma, implied by the result of [CHJT94b], indicates that this is probably not the case.

**Lemma 6.1** Let $X$ and $Y$ be two $M_2$ or $M_3$ objects such that $h_m^r(X) = k_1$, $h_m^r(Y) = k_2$, $h_m^r(X; Y) = k$, and $k > \max\{k_1, k_2\}$. Then, neither $X$ nor $Y$ can be implemented by $CO_{k-1}^\infty \cup R_1^\infty$.

**Proof:** Assume by way of contradiction that $Y$ may be implemented by $CO_{k-1}^\infty \cup R_1^\infty$, then $h_m^r(X; Y) \leq h_m^r(X; CO_{k-1})$. By the result of [CHJT94b], $h_m^r(X; CO_{k-1}) = \max\{h_m^r(X), k-1\}$. Together we get that $h_m^r(X; Y) \leq \max\{h_m^r(X), k-1\} = k - 1$, contradicting the assumption that $h_m^r(X; Y) = k$. Therefore, $Y$ cannot be implemented by $CO_{k-1}^\infty \cup R_1^\infty$. In the same way it can be shown that $X$ cannot be implemented by $CO_{k-1}^\infty \cup R_1^\infty$ as well. □

It turns out that even just finding an $M_2$ or $M_3$ object $X$ such that $h_m^r(X) < k$, but $X$ cannot be implemented by $CO_{k-1}^\infty \cup R_1^\infty$ is a difficult task. Notice that by the universality result, $X$ must be shared by at least $k$ processes. To the best of our knowledge, only one such object is mentioned in the literature [Rac94]. This object is non-deterministic, and it is shared by $2k + 1$ processes.

**Acknowledgments:**

We wish to thank Robert Cori for some very helpful remarks on an earlier version of this paper.

**References**


(observe that in these two runs $P_1$ is not $P^e$). Similarly, $P_3$ will not be able to distinguish between $acb$ and $cab$, if left to run alone. Therefore, $P \not\in \{P_1, P_2, P_3\}$.

- $(1,1,2)$: We show that $P_1$ cannot distinguish between $acb$ and $cab$ when let to run alone after each one of them, until it reaches decision. This implies that $P \not\in \{P_1, P_3\}$. In both $acb$ and $cab$, $P_3 \not\in Q$, therefore $T_3 = 1$, and thus $P_3$ gets the same response on its fourth access to $\mathcal{W}_c$. Since $P_2 \neq P^e$, $P_2$ also gets the same response on its sixth access to $\mathcal{W}_c$. In the same way, it can be shown that $P_1$ cannot distinguish between $bea$ and $cba$ when let to run alone after each one of them, until it reaches decision. This implies that $P \not\in \{P_1, P_3\}$. A similar argument applies to $(1,2,1)$ and $(2,1,1)$.

- $(1,2,2)$: We show that $P_1$ cannot distinguish between $abc$, $bac$ and $cab$ when let to run alone after each one of them, until it reaches decision. This implies that $P \not\in \{P_1, P_3\}$. In $bac$ and $cab$, $P_1 \not\in Q$, therefore, $T_1 = 1$. In $abc$, $P_1 = P_{1(k)} \neq P^e$, therefore, $T_1 = 1$. In all three cases $P_1 \neq P^e$. Therefore, in all three cases $P_1$ gets the same response on its fourth and sixth accesses to $\mathcal{W}_c$. As before, it can be shown that the same argument applies also to $(2,1,2)$ and $(2,2,1)$.

- $(2,2,2)$: Assume without loss of generality, that $P^e \not\in \{P_1, P_3\}$. We show that $P_2$ cannot distinguish between $acb$ and $cab$ when let to run alone after each one of them, until it reaches decision. This implies that $P \not\in \{P_1, P_3\}$. Since in both $acb$ and $cab$ $P_2$ is not the first process which accesses $\mathcal{W}$ three times, $T_2$ is the same for both. Thus, $P_2$ gets the same response on its fourth access to $\mathcal{W}_c$. Since $P_3 \neq P^e$, $P_2$ also gets the same response on its sixth access to $\mathcal{W}_c$. In the same way, it can be shown that $P_2$ cannot distinguish between $bea$ and $cba$ when let to run until alone after each one of them, it reaches decision. This implies that $P \not\in \{P_1, P_3\}$.

This completes the proof of the lemma.

5.3 Proof of the main result

In this subsection, we put the pieces together and prove that $CN(\mathcal{W}^e \cup R^e) < 3$. Assume by way of contradiction, that there exists a wait-free consensus protocol, $Pr$, for three processes over $\mathcal{W}^{\infty} \cup R^{\infty}$. Let the identities of the three processes be $P_i$, $P_j$, and $P_k$. Since $Pr$ is wait-free, there is a number $m$ such that $Pr$ never uses more than $m$ copies of object $\mathcal{W}$. Among all possible protocols, assume that $Pr$ is one for which this number $m$ is minimized. Also, $Pr$ must have a run in which it reaches a critical configuration, $C_{cr}$.

Let $\mathcal{W}_c$ be the critical object in $C_{cr}$. By Lemma 5.7, the state of $\mathcal{W}_c$ in the configuration $C_{cr}$ must be $\epsilon$. But this implies, by the Main Lemma, that there is a consensus protocol for three processes over $\mathcal{W}^e \cup \mathcal{R}^e$, $Pr'$, which uses only $m - 1$ copies of $\mathcal{W}$—a contradiction to the minimality of $m$.

Theorem 5.1 $h^r_m$ is not robust for $M_1$ objects.

Proof: Since we have $CN(\mathcal{W}^{\infty} \cup R^{\infty}) < 3$ and $CN(\mathcal{C}D^m_0 \cup \mathcal{R}^{\infty}) = 3$, but $CN(\mathcal{W}^{\infty} \cup \mathcal{C}D^m_0 \cup \mathcal{R}^{\infty}) = N$, $h^r_m$ is not robust.
5.2 Case 2: $\mathcal{W}_c$’s state in the critical configuration cannot be other than $\epsilon$

In this section, we show that in the critical configuration the state of the critical object, $\mathcal{W}_c$, must be $\epsilon$. This is formalized by the following Lemma:

**Lemma 5.7** Let $Pr$ be a consensus protocol for three processes over $\mathcal{W}_c \cup \mathbb{R}^\infty$. Then there is no critical configuration of $Pr$ in which the state of the critical object, $\mathcal{W}_c$, is not $\epsilon$.

**Proof:** For clarity we assume that the three processes are $P_1$, $P_2$, and $P_3$, and that they are connected to ports 1, 2, and 3 of $\mathcal{W}_c$, respectively (the proof for other processes and ports is identical). Assume that $C_{cr}$ is a critical configuration for $Pr$, and let $\mathcal{W}_c$ be the corresponding critical object. Then there is one process, say $P$, which, if scheduled next, brings the system to a $v$-valent configuration, while if one of the two other processes is scheduled after $C_{cr}$, the system reaches a $(1 - v)$-valent configuration ($v \in \{0, 1\}$). This means that in any run which starts from $C_{cr}$, each non-faulty process will eventually be able to tell whether $P$ was the first to access $\mathcal{W}_c$ after the $C_{cr}$. We reach a contradiction by proving that this last demand cannot hold if the state of $\mathcal{W}_c$ in $C_{cr}$ is not $\epsilon$.

$\mathcal{W}_c$’s state is a sequence of accesses by the $N$ processes (note that $\mathcal{W}_c$ can be initialized to a state in which it “thinks” that some processes already accessed it before the run started). We partition the possible states of $\mathcal{W}_c$ in $C_{cr}$ to classes, according to the number of accesses made to $\mathcal{W}_c$ by $P_1$, $P_2$ and $P_3$. Each class is described by a tuple $(S_1, S_2, S_3)$, where $S_i$, $i \in \{1, 2, 3\}$, denotes the number of times process $P_i$ accessed $\mathcal{W}_c$ before $C_{cr}$. Each one of $\{S_1, S_2, S_3\}$ can either be a number in the range $0-6$, or a sub-range of numbers $x-y$, within that range. For example, $(0,0-2,0-2)$ stands for the set of states in which $P_1$ made no access to $\mathcal{W}_c$ before $C_{cr}$, while $P_2$ and $P_3$ accessed $\mathcal{W}_c$ at most twice before $C_{cr}$.

We start by observing that the only way a process may distinguish between different states of $\mathcal{W}_c$ is by receiving different responses when accessing $\mathcal{W}_c$. Such a response may be a test value $T_i$ received on its fourth access, and for the process $P^c$ (the first process that accessed $\mathcal{W}_c$ for the second time) it may also be a response on its sixth access (which may depend on whether $\mathcal{W}_c$ is stuck). In addition, if in $C_{cr}$ there exists a process which accessed $\mathcal{W}_c$ three or more times, then in this configuration the values of the $T_i$’s are already determined; hence, if we let any of the three processes be the first to access $\mathcal{W}_c$ after $C_{cr}$, and then we let any process $P'$ to run alone, $P$ will receive the same responses in all three cases. Therefore, $P'$ will not be able to decide which process was the first to access $\mathcal{W}_c$, which leads to the desired contradiction.

By the above, we have to check all possible cases of $(S_1, S_2, S_3)$, where $0 \leq S_i \leq 2$. We must show that in each such configuration, there is no process $P \in \{P_1, P_2, P_3\}$ such that every non-faulty process will eventually be able to tell whether $P$ was the first to take a step after $C_{cr}$. Let $a, b$ and $c$ denote the first steps taken after the critical configuration by $P_1$, $P_2$ and $P_3$, respectively.

- $(0,0-2,0-2)$: In this state, $P_1$ is about to make its first access, in step $a$, to $\mathcal{W}_c$. Since the state of $\mathcal{W}_c$ is not $\epsilon$, this step $a$ is ignored by $\mathcal{W}_c$. Therefore, all the states reached by $\mathcal{W}_c$ after $abc$, $bac$ and $bea$ are equivalent, and the same for $acb$, $cab$ and $eba$. Since $abc$, $bac$ and $bea$ are equivalent, $P \notin \{P_1, P_3\}$, and since $acb$, $cab$ and $eba$ are equivalent, $P \notin \{P_1, P_3\}$. The same argument holds for $(0-2,0,0-2)$ and for $(0-2,0-2,0)$.

- $(1,1,1)$: In each of $acb$, $bac$ and $cab$, the values of $(T_1, T_2, T_3)$ are $(0,1,1)$. Thus, if we let $P_1$ run alone after each of $bac$ and $cab$, $P_1$ will not be able to distinguish between them
same before $P^*$'s sixth access to $\mathcal{W}_c$ in $\rho$ and in $\rho'$. Since both in $\rho$ and in $\rho'$ $\mathcal{W}_c$ is stuck while $P^*$ accesses it for the sixth time, $P^*$ gets the same response on its sixth access to $\mathcal{W}_c$, namely, 1. Therefore, $P^*$'s state is also the same after its sixth access to $\mathcal{W}_c$ in $\rho$ and in $\rho'$. After $P^*$ accesses $\mathcal{W}_c$ for the sixth time (both in $\rho$ and in $\rho'$), let $P^*$ run alone until it reaches decision. When $P^*$ reaches decision, it is in the same state in both cases, and thus it reaches the same decision, namely $v$, which is a contradiction.

\textbf{Lemma 5.5} Let $\rho \in \mathcal{R}$, and let $P_i \in \{P_1, P_2, P_3\}$. Then the value $C_{123}$ given by process $P_i$ as part of its input to $\mathcal{W}_c$ in $P_i$'s fifth access to $\mathcal{W}_c$, is a consensus value on $\{T_1, T_2, T_3\}$, and this value is equal for $P_1$, $P_2$, and $P_3$.

\textbf{Proof:} Immediate from Lemma 5.3 and Lemma 5.4.

\textbf{Lemma 5.6} Let $\rho \in \mathcal{R}$, and let $\rho'$ be a prefix of $\rho$ that ends just before all three processes are about to access $\mathcal{W}_c$ in the fourth time. Let $\rho' = \rho_1apbp_3$, where $a$ and $b$ are steps taken by any two processes, in which they access $\mathcal{W}_c$ in the $i$th time, $i \in \{2, 3\}$. Then $\rho'' = \rho_1bp_3ap_3$ is also a prefix of a run in $\mathcal{R}$, and the states of the shared objects in MEM and the states of the processes are same in $\rho'$ and in $\rho''$.

\textbf{Proof:} Immediate, using similar considerations to those used in Lemma 5.1.

We can now prove the Main Lemma. We change $Pr$ to obtain a new wait-free protocol, $Pr'$, which solves the consensus problem for $P_1$, $P_2$ and $P_3$ over $W_{n-1} \cup R^\infty$.

The collection of shared objects used by $Pr'$ is MEM (notice that MEM contains only $m - 1$ objects of type $\mathcal{W}$). The initial configuration of $Pr'$, $C_{init}$, is defined as follows: let $\rho_0$ be an arbitrary (but fixed) run in $\mathcal{R}$, then $C_{init}$ is the configuration reached by $\rho_0$ when $P_1$, $P_2$ and $P_3$ are about to access $\mathcal{W}_c$ for the fourth time. In configuration $C_{init}$, the next step of process $P_i$, $1 \leq i \leq 3$, according to $Pr$, is an access to $\mathcal{W}_c$ in the fourth time (in order to get $T_i$). In $Pr'$, we replace this step by a step in which $P_i$ reads its input (i.e., $T_i$ is replaced by the private input of $P_i$). $P_i$ proceeds in $Pr'$ as it should have in $Pr$, until $P_i$ is instructed by $Pr$ to access $\mathcal{W}_c$ in the fifth time with $(N_3 - 1)$ decision values $C_{ijk}$. Then, instead of accessing $\mathcal{W}_c$, $Pr'$ instructs $P_i$ to decide on $C_{123}$ and halt.

By Lemma 5.4, in each run of $Pr'$, each non-faulty process $P_i$ must reach a stage where in the corresponding run of $Pr$ it would have access $\mathcal{W}$ for the fifth time. Since in $Pr'$ this access is replaced by a step in which $P_i$ decides and halts, protocol $Pr'$ is a wait-free protocol, in which every non-faulty process eventually decides on a binary value.

We prove that $Pr'$ is indeed a consensus protocol for $P_1$, $P_2$ and $P_3$. Assume by way of contradiction that there is a run $\rho'$ of $Pr'$, in which there exists a process which decides on an illegal value, and let $T_1, T_2, T_3$ be the input values in this run. By Lemma 5.6, and the fact that $\mathcal{W}_c$ is not in $\text{MEM}$, we have that $C_{init}$ does not depend on the order in which the processes accessed $\mathcal{W}_c$ in the second and third times. This implies, using Proposition 4.1, that there is a run of $Pr$, which reaches a configuration in which the state of $\text{MEM}$ and the states of the processes are the same as in $C_{init}$, and in which $T_1, T_2, T_3$ are the values returned to $P_1$, $P_2$, and $P_3$ on their fourth accesses to $\mathcal{W}_c$. Hence, there is a run of $Pr$ which starts when the state of the system excluding $\mathcal{W}_c$ is the same as in $C_{init}$, and is identical to $\rho'$. The assumption that in $\rho'$ there is a process which returns an illegal decision value implies that in this latter run of $Pr$ the same process accesses $\mathcal{W}_c$ with an incorrect value of $C_{123}$—contradicting Lemma 5.5.
the step that \( Q_{1,v} \) is about to take in \( C_{vr} \). If \( Q_{1,v} \) takes a step in \( \rho \), then let \( \rho' = \rho_1 a \rho_2 \), and let \( \rho' = a \rho_1 \rho_2 \). If \( Q_{1,v} \) does not take a step in \( \rho \), then let \( \rho' = a \rho \). By Lemma 5.1, \( \rho' \) is also a possible run of \( Pr \), in which the state of \( P_i \) in its decision state is the same as in \( \rho \). Therefore, \( P_i \) reaches the same decision in \( \rho' \) as in \( \rho \), namely, \( v \). This is a contradiction to the fact that the configuration after the first step in \( \rho' \) is \((1 - v)\)-valent.

\[
\text{Given } Pr, \text{ we define a set } \mathcal{R} \text{ of runs, which is the set of all runs of } Pr \text{ which involve processes } P_1, P_2 \text{ and } P_3, \text{ such that for each run } \rho \in \mathcal{R}, \text{ the following conditions hold:}
\]

- the initial configuration of \( \rho \) is the critical configuration \( C_{vr} \).
- Each of the processes \( P_1, P_2 \text{ and } P_3 \) reaches a decision in \( \rho \).
- Each process which is about to access \( \mathbb{W}_c \) in the \( i \)th time, \( i \in \{2, 3, 4, 5\} \), does not take any additional step until the two other processes are about to access \( \mathbb{W}_c \) for the \( i \)th time.
- Each process which accesses \( \mathbb{W}_c \) in the sixth time does not take any additional step until the two other processes access \( \mathbb{W}_c \) six times.

**Lemma 5.3** \( \mathcal{R} \) is not empty, and in each run \( \rho \in \mathcal{R} \) each one of \( P_1, P_2 \text{ and } P_3 \) accesses \( \mathbb{W}_c \) (at least) six times.

**Proof:** In order to show that \( \mathcal{R} \) is not empty, we describe how to schedule the processes \( P_1, P_2 \text{ and } P_3 \), starting from configuration \( C_{vr} \), to obtain a run \( \rho \in \mathcal{R} \). Start by scheduling processes arbitrarily. If a process is about to access \( \mathbb{W}_c \) for the second time, stop scheduling it. Since the protocol is wait-free, each process must reach a decision within a bounded number of its steps, and by Lemma 5.2 this implies that all processes will eventually access \( \mathbb{W}_c \) twice. This procedure can now be repeated for the third, fourth and fifth accesses. A similar procedure can be used for the sixth access. Once all three processes access \( \mathbb{W}_c \) six times, schedule processes arbitrarily, and halt a process which reaches a decision. It is obvious the run obtained by this procedure is indeed in \( \mathcal{R} \).

Using a similar consideration, it is easy to show that in each run \( \rho \in \mathcal{R} \), each one of the three processes accesses \( \mathbb{W}_c \) at least six times.

**Lemma 5.4** Let \( \rho \) be a prefix of a run in \( \mathcal{R} \) which ends when all three processes have accessed \( \mathbb{W}_c \) for the sixth time, then no process may get \( \mathbb{W}_c \) stuck in \( \rho \).

**Proof:** Assume by way of contradiction that there exists a process \( P_i \in \{P_1, P_2, P_3\} \) which gets \( \mathbb{W}_c \) stuck in \( \rho \). Let \( Q_{i,v} \in \{P_1, P_2, P_3\} \) be the process that takes the first step in \( \rho \), and assume that this step leads to a \( v \)-valent configuration. Since \( C_{cr} \) (the initial configuration in \( \rho \)) is bivalent, there exists a process \( Q_{1,v} \in \{P_1, P_2, P_3\} \) that, if scheduled after \( C_{cr} \), would lead to a \((1 - v)\)-valent configuration. Let \( a \) be the first step after \( C_{cr} \) taken by \( Q_{1,v} \), let \( \rho = \rho_1 a \rho_2 \), and let \( \rho' = a \rho_1 \rho_2 \). By Lemma 5.1, \( \rho' \) is also a possible run of \( Pr \).

Process \( P_i \) can only get \( \mathbb{W}_c \) stuck by giving it a wrong value for one of the \( C_{ijk} \)'s on its fifth access, therefore, \( \mathbb{W}_c \) is stuck before any process accesses it for the sixth time. Furthermore, moving \( a \) to be right after \( C_{cr} \) does not change the \( T_i \)'s nor the \( C_{ijk} \)'s, and thus \( P_i \) also gets \( \mathbb{W}_c \) stuck in \( \rho' \). Let \( P^* \) be the first process that accessed \( \mathbb{W}_c \) for the second time in \( \rho \). From the proof of Lemma 5.1, \( P^* \)'s state and the states of all the shared objects excluding \( \mathbb{W}_c \) is the
• $a\rho^j_1$ is a possible run of $Pr$.

• For each $j \leq |\rho_1|$, the state of $\text{MEM}$, and the states of the processes in $P^j$, are the same in $\rho^j_1$ and in $a\rho^j_1$.

**Induction base** $j = 0$: By the definition of $a$, $a$ is a possible run of $Pr$. Since $a$ must be an access to $\hat{W}_c$, the state of $\text{MEM}$ and the states of all processes excluding $P_i$, are the same in the run $\rho^0_1 = \epsilon$ and in the run $\rho^0 a = a$.

**Induction step:** First we show that $a\rho^j_1$ is a run of $Pr$. Let $Q$ be the process that takes the step in $\rho^j_1$. By our assumptions, $Q \neq P_i$, and $Q$ accesses $\hat{W}_c$ at most five times in $\rho^j_1$ and in $a\rho^j_1$. Therefore, $Q \in P^{j-1}$. Since, by the induction assumption, $Q$ is in the same state in $\rho^j_1$ and in $a\rho^j_1$, it will take the same step both in $\rho^j_1$ and in $a\rho^j_1$. Therefore, $a\rho^j_1$ is also a possible run of $Pr$. It remains to show that the induction hypothesis holds for the processes in $P^j$ and for the shared objects in $\text{MEM}$. $Q$ can take one of the following step types in both runs:

1. An access to a shared object in $\text{MEM}$: both $Q$ and the object accessed by it are in the same state in $\rho^j_1$ and in $a\rho^j_1$, and hence $Q$ takes the same step in both runs.

2. One of the first five accesses of $Q$ to $\hat{W}_c$: in both cases, $Q$ gets the same response—none in its first, second, third and fifth accesses, and $T_i$ in its fourth access. $T_i$ is the same in both cases, since the orders in which processes access $\hat{W}_c$ in the second and third times are the same. Therefore, $Q$ moves to the same state in both runs.

3. Sixth access of $Q$ to $\hat{W}_c$: $Q$ may move to different states (in $\rho^j_1$ and in $a\rho^j_1$).

In cases 1 and 2, the state of $Q$ is the same in both runs, and the induction holds for the processes in $P^j = P^{j-1}$. In case 3, the induction holds for $P^j$ since $P^j = P^{j-1} \setminus \{Q\}$, and, by the induction hypothesis for $j - 1$, the states of all the processes in $P^{j-1} \setminus \{Q\}$ are the same in both runs. In cases 2 and 3, the state of $\text{MEM}$ is not affected, and in case 1 it is affected in the same way in both runs. Therefore, by the induction assumption, $\text{MEM}$ is in the same state in both runs.

Let $P^j \setminus \{P_i\}$ be the set of processes that have accessed $\hat{W}_c$ less than six times within $\rho_1 a \rho^j_2$, then $P^j \setminus \{P_i\} \subseteq P^{j-1}$. This fact can be used (in a technique similar to the above) to prove that $a\rho_1 \rho^j_2$ is a run of $Pr$, and that for each $j \leq |\rho_2|$ the state of $\text{MEM}$ and the states of all processes in $P^j$ (including $P_i$) are the same in $a\rho_1 \rho^j_2$ and in $\rho_1 a \rho^j_2$. This implies that $a\rho_1 \rho^j_2$ is a possible run of $Pr$, and completes the proof of the first part of the lemma. The proof of the second part of the lemma, where $P_i$ does not take any step in $\rho$, is similar.

**Lemma 5.2** Let $\rho$ be defined as in Lemma 5.1, and let $P_i$ be a process which accesses $\hat{W}_c$ at most five times in $\rho$, then $P_i$ does not reach a decision state within $\rho$.

**Proof:** Assume by way of contradiction that there exists a process $P_i \in \{P_1, P_2, P_3\}$ which accesses $\hat{W}_c$ at most five times in $\rho$, and reaches a decision state within $\rho$. Let $Q_{\nu} \in \{P_1, P_2, P_3\}$ be the process that takes the first step in $\rho$, and assume that this step leads to a $\nu$-valent configuration. Since $C_{cr}$ (the initial configuration in $\rho$) is critical, there exists a process $Q_{1 - \nu} \in \{P_1, P_2, P_3\}$ that, if scheduled after $C_{cr}$, would lead to a $(1 - \nu)$-valent configuration. Let $a$ be
such that $1 \leq j < k \leq N \land j, k \neq i$, $P_i$ accesses $C_{ij^k}$ with $T_i$ and receives from $C_{ij^k}$ a value $C_{ij^k}$ (which is a consensus value on $\{T_i, T_j, T_k\}$). $P_i$ accesses $\mathcal{W}$ for the fifth time with input $\{C_{ij^k} \mid 1 \leq j < k \leq N \land j, k \neq i\}$, and on its sixth access to $\mathcal{W}$, receives a consensus value on $\{I_1, I_2, ..., I_N\}$. This protocol is wait-free (it has no loops), and it is easy to show that it is correct.

In order to prove that $\mathcal{C.N}(\mathcal{W}^\infty \cup R^\infty) < 3$, we assume by way of contradiction that there exists a wait-free protocol, $Pr$, that solves the consensus problem for three processes, $\{P_1, P_2, P_3\}$, over $\mathcal{W}^\infty \cup R^\infty$. Since $Pr$ is wait-free, there exists a critical run, $\rho_c$, in which the system is in a critical configuration, $C_{cr}$: $C_{cr}$ is a bivalent configuration, but a single step of each of $\{P_1, P_2, P_3\}$ brings the system to a univalent configuration [Her91a]. Since $C_{cr}$ is bivalent, there exists a value $v \in \{0,1\}$ such that two of the three steps bring the system to a $v$-valent configuration, and one will bring it to a $(1-v)$-valent configuration. Also, a standard consideration shows that all the three steps must be applied to the same critical concurrent object. Arguments identical to those in [CIL87, LA87, Her91a] show that the critical object cannot be a read/write register, so it must be a copy of $\mathcal{W}$, to be denoted $\mathcal{W}_c$. We check two cases: either $\mathcal{W}_c$’s state in the critical configuration is $\epsilon$, or it is different from $\epsilon$.

5.1 Case 1: $\mathcal{W}_c$’s state in the critical configuration is $\epsilon$

In this section, we deal with the case where in the critical configuration, $C_{cr}$, defined above, $\mathcal{W}_c$’s state is $\epsilon$, which implies that no process accesses $\mathcal{W}_c$ before $C_{cr}$. Since the responses of $\mathcal{W}$ depend only on the relative order of the processes, we may assume that the identities of the three processes which run $Pr$ are $P_1, P_2$, and $P_3$, and that they are connected to ports 1, 2, and 3 respectively.

Main Lemma: Assume that there is a wait-free protocol, $Pr$, that solves the consensus problem for three processes over $\mathcal{W}^m \cup R^\infty$, for some $m > 0$. Assume further that $Pr$ has a critical configuration in which the state of the critical object $\mathcal{W}_c$ is $\epsilon$. Then there exists a wait-free protocol, $Pr'$, that solves the consensus problem for three processes over $\mathcal{W}^{m-1} \cup R^\infty$.

The Main Lemma is proved using the following lemmas.

Lemma 5.1 Let $\rho$ be any finite run of $Pr$ which starts at the critical configuration $C_{cr}$, such that $\mathcal{W}_c$’s state in $C_{cr}$ is $\epsilon$. Assume that no process takes any step after accessing $\mathcal{W}_c$ six times in $\rho$, and let $P$ be the set of processes that have accessed $\mathcal{W}_c$ at least once and at most five times within $\rho$. Then

1. Let $\rho = \rho_1 \rho_2$, where $a$ is the first step taken by process $P_1$ in $\rho$. Then, $\rho' = a \rho_1 \rho_2$ is also a run of $Pr$, in which the state of each process in $P$ is the same as in $\rho$.

2. If process $P_i$ does not take any step in $\rho$, then $\rho' = a \rho$, where $a$ is the step taken by $P_i$ if it were scheduled immediately after $C_{cr}$, is also a run of $Pr$, in which the state of each process in $P$ is the same as in $\rho$.

Proof: Let $|\rho_1|$ and $|\rho_2|$ be the numbers of steps taken in $\rho_1$ and in $\rho_2$ respectively. Let $\rho_1^j$ and $\rho_2^j$ be the prefixes of length $j$ of $\rho_1$ and $\rho_2$, respectively. Let $P^j$ be the set of processes that have accessed $\mathcal{W}_c$ less than six times within $\rho_1^j$, excluding $P_i$, and let MEM denote the collection of all the objects in the shared memory excluding $\mathcal{W}_c$. Using an induction on $j$, we first show that:

12

We now describe how \( W \) calculates \( T_i \) on the fourth access to port \( i \), for \( 1 \leq i \leq N \). Let \( i_0 \) be the first port in \( W \) that was accessed three times, and assume that \( k \) ports were accessed twice before \( i_0 \) was accessed three times \((1 \leq k \leq N)\). Let \( s(i) \) be the \( i \)th port which was accessed twice before \( i_0 \) was accessed three times, and let \( Q = \{s(1), \ldots, s(k)\} \). Once \( i_0 \) is accessed for the third time, all the \( T_i \)'s, \( 1 \leq i \leq N \), are set. For any port \( j = s(i) \in Q \), \( T_j(= T_{s(i)}) \) is defined in two steps, as follows:

**Step 1**

\[
T'_j = \begin{cases} 
0 & \text{if } s(i) < s(i+1) \land i < k \\
1 & \text{otherwise}
\end{cases}
\]

**Step 2**

\[
T_j = \begin{cases} 
T'_j & \text{if } j \neq i_0 \\
1 - T'_j & \text{otherwise}
\end{cases}
\]

For any port \( j \notin Q \), \( T_j = 1 \).

The above procedure for computing the values \( T_i \) satisfies the following properties:

**Proposition 4.1** Let \((T_1,T_2,T_3)\) be an arbitrary binary vector of size 3, let process \( P_i, i \in \{1,2,3\}, \) be connected to port \( i \) in a given copy of \( W \), and assume that in each run each non-faulty process accesses \( W \) at least four times. Then there is a scheduling of processes \( \{P_1,P_2,P_3\} \) in which process \( P_i, i \in \{1,2,3\}, \) gets output \( T_i \) on its fourth access to \( W \).

**Proof:** Schedule processes \( P_1, P_2 \) and \( P_3 \), such that all three of them access \( W \) twice before any of them accesses \( W \) for the third time. Let \( a,b \) and \( c \) denote the steps in which \( P_1, P_2 \) and \( P_3 \) access \( W \) for the second time, respectively. Then \( abc,acb,bca,bca,ecb \) and \( eca \) denote the possible ways to order the second accesses made by \( \{P_1,P_2,P_3\} \) to \( W \). In \( abc \) we get \( T_1^* = 0, T_2^* = 0 \) and \( T_3^* = 1 \)—for short \( 001 \). In \( acb \) we get \( 011 \), in \( bca \) we get \( 101 \), and in \( eca \) we get \( 111 \). Now, the vector \((T_1,T_2,T_3)\) is obtained by selecting the process that accesses \( W \) first for the third time, causing its entry in \((T_1^*,T_2^*,T_3^*)\) to be flipped. The proof is completed by noting that each binary triplet can be obtained by flipping a bit in one of the vectors \( \{001,011,101,111\} \).

\[\blacksquare\]

## 5 Proof of Main Result

In this section we prove that the conditional consensus object \( W \) satisfies the following properties:

1. \( \mathcal{C}(W) < 3 \).

2. \( \mathcal{C}(W \cup \mathcal{C}_3) = N \), where \( \mathcal{C}_3 \) is the standard consensus object for three processes.

It is easy to show that \( \mathcal{C}(W \cup \mathcal{C}_3) = N \). We describe a wait-free consensus protocol for \( N \) processes over \( W \cup \mathcal{C}_3 \), which uses a single \( W \) object and \( \binom{N}{3} \) \( \mathcal{C}_3 \) objects, tagged \( \{\mathcal{C}_3^{ijk} \mid 1 \leq i < j < k \leq N\} \). Process \( P_i, 1 \leq i \leq N \), which executes the protocol, accesses \( W \) for the first time with \( P_i \)'s input value \( I_i \), and then accesses \( W \) for the second and third times with no input. On its fourth access, \( P_i \) receives from \( W \) a test value \( T_i \). Then, for each \( j \) and \( k \)
In [LH96], Hadzilacos and Lo show that for the model of soft-wired non-deterministic concurrent objects, $h^r_m$ is non-robust. That is, they generalize Schenk's results in two ways: They do not assume objects which are infinite-branching, and they use the usual definition of wait-freedom. They introduce two non-deterministic objects, negation and booster, such that $h^r_m(\text{negation}) = 1$, $h^r_m(\text{booster}) = 1$, but $h^r_m(\text{negation,booster}) = 2$. In Part II of this paper, they generalize this result by introducing a generalization of the above objects, called negation$_n$ and booster$_n$, for which $h^r_m(\text{negation}_n) = 1$, $h^r_m(\text{booster}_n) = 1$, but $h^r_m(\text{negation}_n,\text{booster}_n) = n$.

4 The Conditional Consensus Object

In this section, we introduce the conditional consensus object, $\hat{W}$, which is an $M_1$ object for $N \geq 3$ processes, $P_1, \ldots, P_N$. In the following section we use $\hat{W}$ to prove our main result.

An operation applied on $\hat{W}$ can be described as a pair $(i, v)$, where $i$ is a port number (in which the operation is applied), and $v$ is an input value. Each state of $\hat{W}$ is a sequence of operations applied to it: whenever an operation is applied on $\hat{W}$, the state of $\hat{W}$ is extended by this operation. Thus, if the initial state of (a copy of) $\hat{W}$ in a given run is $\epsilon$, then the state of $\hat{W}$ in this run is exactly the sequence of operations performed on $\hat{W}$. Note that $\hat{W}$ "knows" from its state exactly how many times it was accessed from each port. $\hat{W}$'s response to a given operation depends on $\hat{W}$'s state, the port number used and on the input applied to $\hat{W}$. $\hat{W}$ reacts to an access on port $i$ according to the number of times $\hat{W}$ was accessed on that port so far, as follows:

1. First access:
   - Input: $I_i$. If the state of $\hat{W}$ is $\epsilon$, $\hat{W}$ sets its decision variable, $D$, according to $I_i$—if $I_i = 0$, $D$ is set to 0, otherwise $D$ is set to 1. If the state of $\hat{W}$ is not $\epsilon$, $\hat{W}$ ignores $I_i$.
   - Output: $\text{nil}$.


5. Fifth access:
   - Input: $\{C_{ijk} \mid 1 \leq j < k \leq N \land j, k \neq i\}$, where $C_{ijk}$ is a consensus on $\{T_i, T_j, T_k\}$. For $C_{ijk}$ to be legal, it must be equal to one of $\{T_i, T_j, T_k\}$, and the value of $C_{ijk}$ given in ports $i$, $j$, and $k$ must be the same. If $\hat{W}$ is accessed with an illegal input, $\hat{W}$ is said to be stuck.
   - Output: $\text{nil}$.

6. Sixth access:
   - Input: ignored.
   - Output: A consensus on $\{I_1, I_2, \ldots, I_N\}$, which is the value of $\hat{W}$'s decision variable, $D$. If, however, $\hat{W}$ is stuck, and $i$ is the first port in which $\hat{W}$ is accessed twice, then $\hat{W}$ outputs 1.
booster object is described in detail in [CHJT94a]. It is stated in [CHJT94a] that booster can be generalized, such that for all \( n \geq 3 \), one can obtain an \( n \)-ported object \( T_n \) such that \( h^r_m(T_n) = 1 \) and \( h^r_m(T_n, \text{CO}_2) = n \). Our result shows that this latter result holds also for the class of deterministic \( M_1 \) objects.

3.2 Results of Peterson, Bazzi and Neiger

In [PBN94], Peterson, Bazzi and Neiger study \( n \)-ported objects. They show that for that class, level \( n \) of the hierarchy is robust. Specifically, they present a result which implies that the following conditions are equivalent for a hard-wired \( n \)-ported concurrent object \( X \):

(a) \( h^r_m(X) < n \).

(b) For any \( n \)-ported object \( Y \) with \( h^r_m(Y) = n - 1 \), \( X \) can be implemented by \( Y^\infty \cup R^\infty \).

((b) ⇒ (a)) follows from the definition of \( h^r_m \), by letting \( Y = \text{CO}_{n-1} \); the main contribution in [PBN94] is in proving (a) ⇒ (b)). There is no similar result for \( n \)-ported objects with consensus number at most \( k \), for any fixed \( k < n - 1 \). In particular, it is stated in [PBN94] that there exists an \( n \)-ported concurrent object \( X \), such that \( h^r_m(X) < n - 2 \), but \( X \) cannot be implemented by some of the \( n \)-ported objects with consensus number \( n - 2 \).

Our result implies that the result of [PBN94] does not hold for deterministic (and non-deterministic) \( M_1 \) objects. In particular, for \( n > 3 \), our object \( \check{W} \) satisfies \( h^r_m(\check{W}) < 3 < n \), but \( \check{W} \) cannot be implemented by \( \text{CO}_{n-1}^\infty \cup R^\infty \), since this would imply that \( n - 1 = h^r_m(\text{CO}_{n-1}^\infty) = h^r_m(\text{CO}_{n-1} \cup \check{W}) = n \), a contradiction. Jayanti shows in [Jay95] that, assuming \( M_2 \) objects, the result of [PBN94] implies that \( h^r_m \) is robust (for all levels).

In [BNP94] it is shown that for deterministic objects, \( h^r_m \) and \( h^r_r \) are the same (i.e. for each object \( X \), \( h^r_m(X) = h^r_r(X) \)), which implies that \( h^r_m \) is robust iff \( h^r_r \) is robust. It is then shown that Jayanti’s use (in [Jay93]) of non-deterministic objects to show that \( h^r_m \) is not robust is a must. Finally, it is claimed that since it was proved in [PBN94] that \( h^r_m \) is robust for deterministic objects, this implies that (for deterministic objects) \( h^r_m \) is robust as well.

3.3 Results of Borowsky, Gafni and Afek

The main result of [BGA94] states that any task over \( m \leq 2n \) processes which is solvable using only objects \( X \) with \( h^r_m(X) < n + 1 \), is also solvable by \( \text{CO}_n^\infty \cup R^\infty \). In particular, any collection of objects whose consensus number is at most \( n \) cannot implement \( n + 1 \) consensus, which means that \( h^r_m \) is robust. Our result implies that this result of [BGA94] does not hold for \( M_1 \) objects.

3.4 Results of Hadzilacos, Lou and Schenk

In [Sch96], Schenk shows that for the model of soft-wired infinite-branching non-deterministic concurrent objects, \( h^r_m \) is not robust. An infinite-branching non-deterministic object, allows to make a non-deterministic choice from an infinite set. Furthermore, Schenk uses a definition of wait-freedom which is slightly weaker than the common one: an algorithm is wait-free if, for any input, there is a bound on the number of steps a process may take before it halts (that is, the bound may depend on the input). See [BM96] for detailed comparison of these two definitions. Schenk introduces two objects, Key and Lock, such that \( h^r_m(\text{Key}) = 1 \),
with a single port, $\mathcal{X}'$, where $S_{\mathcal{X}} = S_\mathcal{X}, R_{\mathcal{X}} = R_\mathcal{X}, OP_{\mathcal{X}} = OP_\mathcal{X}$ and $\delta_{\mathcal{X}} = \delta_\mathcal{X}$. An operation $OP$ executed by process $P_i$ on port $i$ of $\mathcal{X}$ is emulated on $\mathcal{X}'$ by connecting $P_i$ to the single port of $\mathcal{X}'$, executing the same operation $OP$ on that port, and finally disconnecting $P_i$ from the port. Emulating $\mathcal{X}'$ by $\mathcal{X}$ is obvious.

Lemma 2.3 The class of oblivious $M_3$ objects is equivalent to the class of non-oblivious $M_3$ objects with a single port, $M_3^1$.

2.6 Consensus Protocols and Consensus Numbers

A wait-free consensus protocol (in short a “consensus protocol”) for $n$ processes over a given set of concurrent objects, $\mathcal{S}$, is a protocol for $n$ processes over $\mathcal{S}$, defined as follows: each process receives an input, and in every possible run of the protocol, every process which makes infinitely many steps eventually decides on some input value, such that all the processes which decide, decide on the same value.

A run of a consensus protocol is said to be in a bivalent configuration, if the eventual decision value is not yet determined. A run is in a $0$-valent ($1$-valent) configuration, if in all possible extensions of this run, the eventual decision value of the processes is $0$ ($1$). A configuration which is $0$-valent or $1$-valent is said to be univalent. Finally, a univalent configuration is decisive if some process had actually decided in this run. See [FLP85, LA87, BMZ90, TKM94, LM95] for further exposition of these terms, and of consensus protocols in general.

The consensus number of a set $\mathcal{S}$ of concurrent objects, denoted $CN(\mathcal{S})$, is defined as follows: if there is an integer $n$ such that there is a consensus protocol for $n$ processes over $\mathcal{S}$, and there is no consensus protocol for $n+1$ processes over $\mathcal{S}$, then $CN(\mathcal{S}) = n$. If for every $n$ there is such a protocol, then $CN(\mathcal{S})$ is unbounded. If there is a consensus protocol for an infinite set of processes over $\mathcal{S}$, then $CN(\mathcal{S}) = \infty$. For an object $\mathcal{X}$ and $n \in \{1, 2, 3, \ldots\} \cup \{\infty\}$, $\mathcal{X}^n$ denotes a set consisting of $n$ objects, each of which is isomorphic to $\mathcal{X}$, and $CN(\mathcal{X})$ denotes $CN(\{\mathcal{X}\})$.

3 Related Work

Following [Jay93], the robustness of $h^*_m$ with respect to various classes of concurrent objects was studied in few papers. We survey below the results obtained in [BGA94, CHJT94a, CHJT94b, PBN94, BNP94, LH96, Sch96], and compare them with our result. We observe that in these papers there is no explicit distinction between the $M_1$ and $M_3$ models. This distinction was introduced later in [Jay95], following an earlier version of this paper.

3.1 Results of Chandra, Hadzilacos, Jayanti and Toueg

Two results from [CHJT94b] concern the robustness of $h^*_m$. The first result hold for $M_2$ and $M_3$ objects, and it states that for any set $\mathcal{S}$ of objects (either deterministic or non-deterministic), $h^*_m(\mathcal{S}, COH) \geq N+1 \Rightarrow h^*_m(\mathcal{S}, COH-1) \geq N$. By recursively applying this result, it implies that $h^*_m(\mathcal{S}, COH-1) \geq N \Rightarrow h^*_m(\mathcal{S}) \geq N$, which in turn implies that $CN(\mathcal{S}, COH) = \max\{h^*_m(\mathcal{S}), N\}$.

The second result of [CHJT94b] shows that $h^*_m$ is not robust for the class of non-deterministic $M_1$ objects. This is proved by presenting a non-deterministic $M_1$ object called booster, which has the following property: $h^*_m(\text{booster}) = 1$, but $h^*_m(\text{booster}, CO_2) = 3$. I.e., the booster object shows that levels 2 and 3 of the $h^*_m$ hierarchy collapse, in some precise sense. The
execution in which implemented operations do not overlap. Let \( E \) be a sequential execution, let \( OPE \) be the sequence of implemented operations in \( E \), and let \( RE \) be the corresponding sequence of responses. Let \( R \) be the sequence of responses returned by \( A \) for the same sequence of operations \( OPE \), then \( E \) is correct, if \( RE = R \).

An execution \( E \) is linearizable \([HW90]\) with respect to an implementation \((I, I, R)\), if it is possible to define a linearization point for each implemented operation, such that the following conditions hold:

1. The linearization point of each implemented operation is an event within its execution interval.

2. Let \( S \) be the sequence of operations defined by the order of the linearization points, and let \( E_{seq} \) be a sequential execution of the operations in \( S \). Then the sequence of responses in \( E \) and in \( E_{seq} \) are the same.

An implementation of a concurrent object is linearizable if all its executions are linearizable. An implementation of a concurrent object is correct if it is linearizable, and each of its sequential executions is correct.

A concurrent object implementation is non-blocking if it always guarantees that some process completes an operation within a bounded number of steps. A concurrent object implementation is wait-free if it always guarantees that each process completes an operation within a bounded number of its own steps.

### 2.5 Redundancy of Oblivious types

In this subsection, we show that the following classes are equivalent: oblivious \( M_1 \), oblivious \( M_2 \), oblivious \( M_3 \) and non-oblivious \( M_3 \). Since all three oblivious classes are equivalent to the non-oblivious \( M_3 \) class, it suffices to consider only non-oblivious classes. Thus, in the rest of the paper we will refer only to the three classes of non-oblivious objects: \( M_1 \), \( M_2 \) and \( M_3 \).

We start by formally defining what does it mean for two classes of objects to be equivalent. A concurrent objects \( X \) can be emulated by a concurrent objects \( Y \), if \( X \) can be implemented using a single copy of \( Y \), such that each one of the operations defined on \( X \) may be replaced by a single operation of \( Y \). Two concurrent objects \( X \) and \( Y \) are said to be equivalent if \( X \) and \( Y \) may emulate each other. Two classes of objects \( X \) and \( Y \) are said to be equivalent if for each object \( X \in X \) there exists an object \( Y \in Y \) such that \( X \) and \( Y \) are equivalent, and vice-versa.

**Lemma 2.1** The class of non-oblivious \( M_3 \) objects is equivalent to the subclass of non-oblivious \( M_3 \) objects with only a single port, \( M_3^1 \).

**Proof:** Since \( M_3^1 \subseteq M_3 \), it remains to show that \( M_3 \subseteq M_3^1 \). Given an object \( X \in M_3 \) with \( n > 1 \) ports, it can be emulated by an object \( X' \in M_3^1 \) (with a single port): \( S_{X'} = S_X, R_{X'} = R_X, OP_{X'} = OP_X \times \{1...n\} \) and \( \delta_{X'}(s, (op, i)) = \delta_X(s, op, i) \).

**Lemma 2.2** The class of oblivious hard-wired objects (\( M_1 \) or \( M_2 \)) is equivalent to the class of non-oblivious \( M_3 \) objects.

**Proof:** Notice that the transition function \( \delta \) for both classes is the same: \( \delta : S \times OP \rightarrow S \times R \). Given an oblivious hard-wired object \( X \), it can be emulated by non-oblivious soft-wired object
input values define a mapping from schedules into executions. Notice that the initial states of
the objects and the (initial) port connections do not depend on the input values.

A decision task for \( n \) processes is a tuple \((IN, OUT, D)\), where:

- \( IN \) is a set of legal input vectors. An input vector \( v \in IN \) is of the form \( v = (v_1, \ldots, v_n) \),
  where \( v_i \) is the input value of process \( P_i \).
- \( OUT \) is a set of output vectors. An output vector \( d \in OUT \) is of the form \( d = (d_1, \ldots, d_n) \),
  where \( d_i \) is the decision value of process \( P_i \).

\( D : IN \rightarrow 2^{OUT} \) is a mapping from input vectors into sets of output vectors.

\[ d = (d_1, \ldots, d_n) \] is a partial output vector if there exists an output vector \( d' = (d'_1, \ldots, d'_n) \in OUT \)
such that for each \( i \), \( 1 \leq i \leq n \), either \( d_i = d'_i \) or \( d_i = - \). In this case we say that \( d \) is extendible to \( d' \).

Let \( Pr = (\mathcal{P}, M, I_M) \) be a (wait-free) protocol, and let \( T = (IN, OUT, D) \) be a task. \( Pr \)
 solves \( T \) if for each input vector \( v = (v_1, \ldots, v_n) \in IN \), and for each run in which the input to
process \( P_i \) is \( v_i \), the output vector composed of the decision values of the processes is extendible
to a vector in \( D(v) \).

2.4 Implementations

An implementation of a concurrent object \( A \) by a set of concurrent objects \( I \), replaces each
operation of \( A \) by a routine (or a set of routines) over \( I \). \( A \) is called the implemented object,
and the objects in \( I \) are called the implementing objects. Intuitively, the implementation is
considered correct if the processes in the system cannot distinguish between the input/output
behavior of \( A \) and its implementation. This is formalized by the linearity correctness
condition [HW90]. Note that the implementing routines may depend on the initial state of the
implemented object. Therefore, a formal definition of implementation must consider also the
initial state of the implemented object.

Let \( A \) be an \( n \)-ported concurrent object, and let \( OP_A = \{op_1, op_2, \ldots\} \) be the set of operations
defined on \( A \). For a given state \( s \) of \( A \), an implementation of \( A \) from an initial state \( s \) is a
triplet \((I, I_s, R)\), where \( I \) is a set of (implementing) objects, \( I_s \) is a set of initial states for the
objects in \( I \), and \( R = \{r_1, r_1^1, \ldots, r_1^n, \ldots\} \) is a set of routines, which use only operations
defined over objects in \( I \), and satisfy the correctness conditions defined below. Routine \( r_i^j \) is the
routine executed by a process connected to port \( j \) when it executes the operation \( op_i \in OP_A \).

A set of objects \( I \) implements an object \( A \) if there exists an implementation of \( A \) from \( I \) for
every initial state of \( A \).

If \( I \) implements \( A \), then any protocol \( P_r \) over a set of objects \( S \cup \{A\} \), \( Pr = (\mathcal{P}, S \cup \{A\}, I_s \cup I(A)) \),
 can be replaced by an equivalent protocol \( P_r' \) over \( S \cup I \), \( Pr' = (\mathcal{P}', S \cup I, I_s \cup I(T)) \), where \( \mathcal{P}' \) is obtained from \( \mathcal{P} \) by replacing each state in \( \mathcal{P} \) in which process \( P \) applies the operation
\( op_i \) on port \( j \) of \( A \) by the corresponding routine \( r_i^j \), as defined by the implementation.

Let \( P \) be a process which applies an implemented operation \( op \) of an implemented object
\( A \). Let \( e_i \) be the first atomic operation (an operation defined on the implementing objects)
executed by process \( P \) after \( P \) enters the routine \( r \) for implementing \( op \), and let \( e_j \) be the
last atomic operation executed by \( P \) before \( P \) exits \( r \). The execution interval of \( op \) is the
subsequence of events \( (e_i, \ldots, e_j) \) of the system execution. Two implemented operations are
said to be overlapping if their execution intervals intersect. A sequential execution is an
to its current state and the value returned by the atomic operation \( op \). A process \( P \) may have two special atomic operations: a read-once \textit{input} operation, in which \( P \) reads an input value (given by some external device), and a write-once \textit{decision} operation, in which it writes a decision value to a specified register. In dynamic and soft-wired connection schemes, the connection (and release) to a port is considered as part of the corresponding atomic step.

Formally, a sequential process is defined by a 4-tuple \( (S, s_0, O, I, \delta) \), where:

\( S \) is a (possibly infinite) set of \textit{states}.  
\( s_0 \in S \) is an \textit{initial state}.  
\( O \) is a (possibly infinite) set of \textit{output values}. An output value is a triplet \( (X, p, op) \), where \( X \) is a concurrent object, \( p \) is a port in \( X \), and \( op \) is an operation defined on \( X \).  
\( I \) is a (possibly infinite) set of \textit{input values} (results from operations on the concurrent objects of the system).  
\( \delta : S \times I \rightarrow S \times O \) is a transition function, which specifies for each state of the process and for each input what is the next state and what is the output value returned by the process.

When we consider a system with a set of processes \( P \), we use the term \textit{routine} to denote a program which is used by the processes in \( P \). Thus, a routine is defined in the same way a process is.

### 2.3 Concurrent Systems, Protocols and Tasks

A \textit{concurrent system} is a pair \( (P, M) \), where \( P \) is a set of processes and \( M \) is a set of concurrent objects. A \textit{protocol} for \( P \) over \( M \) is a triplet \( (P, M, I_M) \), where \( I_M \) defines the initial state of each of the objects in \( M \). That is, a protocol can be thought of as a collection of programs—one program for each process in \( P \), plus assignments of initial states to each of the objects in \( M \). The initial state of an object may be any one of the object’s states. If the system includes hard-wired objects with static port connection scheme, \( I_M \) also specifies which process is connected to which port in these objects.

Let \( CS = (P, M) \) be a concurrent system with set of processes \( P = (P_1, \ldots, P_n) \) and set of concurrent objects \( M = (x_1, \ldots, x_m) \). A \textit{configuration} of \( CS \) is a vector \( C = (q_1, \ldots, q_n, v_1, \ldots, v_l) \) where \( q_i, 1 \leq i \leq n, \) is a state of process \( P_i \) and \( v_j, 1 \leq j \leq m, \) is a state of object \( x_j \). An \textit{initial configuration} is a configuration in which all processes are in their (local) initial states and all objects are in their (local) initial states (according to the protocol). A computation in \( CS \) proceeds through a sequence of configurations.

A \textit{schedule} \( S \) is a sequence of process names. An \textit{execution segment} of \( CS \) is a (finite or infinite) sequence with the following form: \( C_0, \phi_0, C_1, \phi_1, C_2, \phi_2, \ldots \), where \( C_k, k = 0, 1, 2, \ldots \), are configurations, and \( \phi_k, k = 0, 1, 2, \ldots \), are process names. Furthermore, for all \( k \), \( C_{k+1} \) is the configuration obtained by applying the transition function of process \( \phi_k \) on the previous configuration \( C_k \). An \textit{execution} in the system is an execution segment \( C_0, \phi_0, C_1, \phi_1, C_2, \phi_2, \ldots \), where \( C_0 \) is the initial configuration.

Notice that if processes are deterministic, then the execution is uniquely determined by the initial configuration and the schedule. A \textit{run} of a deterministic protocol \( Pr \) is an execution according to \( Pr \) with inputs, that is, it is a triplet \( (Pr, S, IN) \), where \( S \) is a schedule, and \( IN \) is the set of values returned by the read-once \textit{input} operation of each process. A protocol plus
3. At any given moment, at most one process may be connected to a given port.

All connection schemes considered in this paper satisfy rule 3, which is derived by the atomicity assumption. Objects which satisfy rule 1 are called hard-wired objects [BGA94]. Notice that in hard-wired objects, a process may be connected to different ports in different copies of the same object. Jayanti in [Jay95] defines three connection schemes, $M_1$, $M_2$ and $M_3$, as follows: In $M_1$, all three rules apply. In $M_2$, only rules 1 and 3 apply, that is, a process may be connected to more than one port in a given object. In $M_3$, only rule 3 applies, which means that a process may be disconnected from a port and then be reconnected to (another) port in run time. $M_1$ and $M_2$ characterize hard-wired objects, while $M_3$ characterize soft-wired objects. It is possible to further sub-divide hard-wired objects according to the time in which processes are allowed to be connected to objects: a static connection model allows processes to be connected to objects only at the protocol initialization stage, while a dynamic connection model allows processes to be connected to objects also while the protocol is executed (in run time).

Hard-wired objects model objects which are shared by processors using hardware that can distinguish among the different processors, while soft-wired objects model objects which are shared either by processes or by indistinguishable processors. Although it is more accurate to use the term processors when relating to hard-wired objects, we will use the term processes for both hard-wired and soft-wired objects.

Examples: A (soft-wired) read/write register, over a set of values $V$, is defined by $(n, S_{reg}, OP_{reg}, R_{reg}, \delta_{reg})$, where $S_{reg} = V$, $R_{reg} = V \cup \{ok\}$, and $OP_{reg}$ consists of a read operation and of $|V|$ write operations (one write operation for each value in $V$). Let $v, v' \in V$, then $\delta_{reg}(v, read) = (v, v)$, and $\delta_{reg}(v, write) = (v', ok)$.

A natural example of an $M_1$ object, is an object which supports the operations Load Linked (LL) and Store Conditional (SC). Informally, $LL(x)$ reads the value of a concurrent variable $x$, such that $x$ may be subsequently used in combination with a SC operation. $SC(x, a)$ writes the value $a$ to $x$ and returns SUCCESS if no other successful SC($x$) was executed since the last LL($x$) operation executed by the process, and returns FAILURE otherwise. Formally, a (hard-wired) LL-SC object, over a set of values $V$, which is shared by $n$ processors, is defined by $(n, S_{LL-SC}, OP_{LL-SC}, R_{LL-SC}, \delta_{LL-SC})$, where $S_{LL-SC} = V \times 2^{\{1..n\}}$, $R_{LL-SC} = V \cup V \times \{SUCCESS, FAILURE\}$, and $OP_{LL-SC}$ consists of a LL operation and of $|V|$ SC operations (one SC operation for each value in $V$). Let $v, v' \in V$, $p \in \{1..n\}$, $P \subseteq \{1..n\}$, then $\delta_{LL-SC}(p, (v, P), LL) = ((v, P \cup \{p\}), v)$, and

$$\delta_{LL-SC}(p, (v, P), SC_{v'}) = \begin{cases} (v', -), SUCCESS & p \in P \\ (v, P), FAILURE & p \notin P \end{cases}$$

2.2 Processes

Informally, a process is a thread of control. It can be identified with the program it is running, together with the local variables used by the program. Each such program consists of internal operations, of port connections and disconnections, and of atomic steps on the concurrent objects of the system. Once a process is scheduled it performs a single operation on a concurrent object according to its program. The process may then execute arbitrarily many internal operations. A process $P$ accesses a concurrent object $X$ by performing an atomic step, which consists of two phases: activating an atomic operation $op$ on $X$, and changing $P$’s state according
4. Finally, each process accesses $W$ in order to receive the decision value $D$ that $W$ stores at phase 1. However, if $W$ receives an incorrect value, $C_{ijk}$, in phase 3, $W$ may return a wrong value, instead of $D$.

The tricky parts in the construction and in the proof are in making the test values unpredictable by the processes, and in making $W$'s responses to the processes reveal as little information as possible on $W$'s internal state. We are then able to show that processes which try to reach a consensus over $W^\infty \cup R^\infty$ must provide to one of the copies of $W$ correct consensus values for all triplets of test values, without using this copy of $W$. We use this last fact to show that the assumption that there is a consensus protocol for three processes over $W^\infty \cup R^\infty$ for any number $m$ leads to a contradiction, which implies our result.

2 Definitions and Notations

A concurrent system consists of a collection of processes and of a shared memory. The shared memory is modeled by a (possibly infinite) set of deterministic or non-deterministic concurrent objects. A concurrent object is a basic memory unit, which enables a given set of processes to perform predefined operations on it. Processes are modeled by infinite state machines, and they are asynchronous—there is no global clock timing them. Each process runs at a different speed, and might be subject to arbitrarily long delays. In particular, a process cannot tell whether another process is halted or is running very slowly. The processes communicate by applying operations on the concurrent objects. We assume that the concurrent objects are atomic, hence all operations applied on the objects are totally ordered in time.

2.1 Concurrent Objects

Formally, a (deterministic) concurrent object is defined by a 5-tuple $(n, S, OP, R, \delta)$, where: $n$ is the number of ports, $S$ is a (possibly infinite) set of states, $OP$ is a (possibly infinite) set of operations (composed of a function name and parameters), $R$ is a (possibly infinite) set of responses (output values) which can be returned by the object, and $\delta : \{1..n\} \times S \times OP \rightarrow S \times R$ is a transition function, which specifies for each port in $\{1..n\}$, each state of the object, and each operation applied to it, what is the next state and what is the output value returned by this operation. A non-deterministic object extends the transition function to a relation: $\delta : \{1..n\} \times S \times OP \rightarrow 2^{S \times R}$.

Some papers consider also oblivious objects, which are special type of a concurrent object, in which the transition function $\delta$ does not depend on the port being used: $\delta : S \times OP \rightarrow S \times R$. As we show later, oblivious objects can be modeled as (non-oblivious) objects with appropriate connection schemes (which we describe next).

In order to completely specify a concurrent object, one should also specify the rules by which a process may connect or disconnect from the object’s ports. For this, we augment the definition of a concurrent object to be a pair $((n, S, R, OP, \delta), M)$, where $M$ is a port-connection scheme applicable to the object. Following [Jay95], we classify connections schemes by whether or not they satisfy the following rules:

1. Once a process is connected to a port, it cannot disconnect.
2. Each process may be connected to at most one port.
implement a strong object from a set of weak objects. Jayanti also relates to the notion of power captured by \( h_m^r \). If \( h_m^r(X) > h_m^r(Y) \), \( X \) is more powerful than \( Y \) in the following sense: \( X \) is universal for a larger number of processes than \( Y \), that is, the maximum number of processes for which \( X \) can solve any synchronization problem is larger than that of \( Y \). This does not mean that \( X \) can solve any problem that \( Y \) can. In particular, [Rac94] shows just the opposite.

The robustness of \( h_m^r \) can be studied with respect to different models, e.g., one can either allow non-determinism or not, either allow hard-wired objects or not, and use different ways for processes to be connected to the objects (we discuss the different models of concurrent objects in detail in the next section). The question of the robustness of \( h_m^r \) can be now formulated as follows: given a class \( \mathcal{X} \) of of concurrent objects, is \( h_m^r \) robust with respect to \( \mathcal{X} \)? Formally, \( h_m^r \) is robust with respect to \( \mathcal{X} \) if for each subset of objects \( \{X_1, \ldots, X_n\} \subseteq \mathcal{X} \), \( h_m^r(X_1, \ldots, X_n) = \max\{h_m^r(X_1), \ldots, h_m^r(X_n)\} \). It turns out that different works consider different classes of objects, which lead to different results. We discuss these works in detail in the sequel.

This paper shows for the first time that the \( h_m^r \) hierarchy is not robust for deterministic hard-wired objects. We introduce a new deterministic concurrent object for \( N \geq 3 \) processes, the conditional consensus object, denoted as \( \mathcal{W} \), such that \( \mathcal{C} \mathcal{N}(\mathcal{W}^\infty \cup \mathcal{R}^\infty) < 3 \), but \( \mathcal{C} \mathcal{N}(\mathcal{W} \cup \mathcal{C} \mathcal{O}_3^N) = N \) (\( \mathcal{C} \mathcal{O}_3 \) is the standard consensus object for three processes). The conditional consensus object \( \mathcal{W} \) is hard-wired, in the sense that each process \( P \) can access it using a single fixed port (though \( P \) can use different ports for different copies of \( \mathcal{W} \)). Examples of hard-wired objects found in the literature include single-writer multi-reader registers [Lam86], and Load Linked and Store Conditional [Her91b]. Our work is closely related to [CM94], where similar ideas are used to present a different aspect of the non-robustness of the consensus number hierarchies.

### 1.1 An Outline of the Construction

The intuition behind the conditional consensus object, \( \mathcal{W} \), is as follows. Each copy of \( \mathcal{W} \) is similar to the hard-wired version of the standard consensus object for \( N \) processes, \( \mathcal{C} \mathcal{O}_N \),1 with the following modification: each copy of \( \mathcal{W} \) responds with the right consensus value only after it gets a “proof” that the system has the power to reach a consensus for three processes without using this specific copy of \( \mathcal{W} \). Specifically, \( \mathcal{W} \) has four phases of operations, during which each process accesses it six times:

1. \( \mathcal{W} \) receives the input values of the processes that access it, and stores the first input as its decision value, \( D \).

2. \( \mathcal{W} \) gives each process \( P_i \) a test value, \( T_i \). In order to make the test values unpredictable by the processes, \( \mathcal{W} \) decides on the test values according to the order in which the processes access it (this order is unknown to the processes). In this way, we use the non-determinism of the scheduler to make the object look non-deterministic to the processes. As will be explained later on, this phase requires each process to access \( \mathcal{W} \) three times.

3. Each process \( P_i \) accesses \( \mathcal{W} \) with \( \binom{N-1}{2} \) decision values, one for each triplet of processes \( \{P_i, P_j, P_k\} \) which includes \( P_i \). The decision value, \( C_{ijk} \), for each such triplet, must be a consensus value consistent with the test values, \( \{T_i, T_j, T_k\} \), given by \( \mathcal{W} \) to the processes in that triplet.

---

1 We assume that the “standard” consensus object \( \mathcal{C} \mathcal{O}_N \) is soft-wired.
1 Introduction

Research on wait-free asynchronous shared memory systems has led to the definition of some fundamental concurrent objects, which are used in the implementation of shared data structures, and for synchronization tasks among concurrent processes. This led to the question of the power of different objects—more powerful objects being able to perform harder tasks than less powerful ones.

An important progress in this direction was achieved by Herlihy in [Her91a], where he used consensus protocols [FLP85] for the classification of objects, and introduced the notion of the consensus number of a concurrent object. The consensus number of a concurrent object $X$, defined in [Her91a], is the maximal number $k$ for which a wait-free consensus protocol involving $k$ processes, using a single copy of $X$ and any number of read/write registers, exists. If there is no such maximum, then the consensus number of $X$ is $\infty$.

In [Her91a], Herlihy computes the consensus number of many fundamental objects (e.g., read/write registers, stacks, queues, etc.), and shows the existence of a hierarchy of objects with respect to wait-free implementation. The hierarchy maps the objects into levels, and Herlihy shows that an object $X$ in level $N$ of that hierarchy, together with read/write registers, can be used to construct a wait-free implementation of any concurrent object $Y$ shared by $N$ processes. This last property of Herlihy’s hierarchy is called the universality property, and it formulates the notion of power conveyed by this hierarchy. A hierarchy of concurrent objects which has the universality property is called a wait-free hierarchy [Jay93].

In this paper, as in a previous paper [CM94], we use a slightly more general notation for the term consensus number, which holds for any collection of shared objects. Let the consensus number of a given set $S$ of shared objects, denoted $CN(S)$, be the maximum number $n$ such that there is a wait-free consensus protocol for $n$ distinct processes, which communicate only by accessing objects in $S$. Also, for an object $X$, let $X^n$ denote a collection of $n$ copies of $X$, and let $\mathcal{X}$ denote $\mathcal{X}^1 = \{x\}$. Note that the consensus number of an object $X$ as defined in [Her91a] is $CN(X \cup R^\infty)$, where $R^\infty$ denotes an infinite set of read/write registers.

For most “standard” concurrent objects $X$ and $Y$, it holds that $CN(X \cup R^\infty) = CN(X^\infty \cup R^\infty)$, and $CN(X \cup Y \cup R^\infty) = \max\{CN(X \cup R^\infty), CN(Y \cup R^\infty)\}$. This led to the belief that an object with a large consensus number cannot be implemented using any combination of objects, each having a smaller consensus number. Informally, such a property means that the consensus number is a robust property. The question of robustness was presented and formally defined by Jayanti in [Jay93].

In [Jay93], Jayanti suggests four ways for classifying objects using consensus numbers. An object $X$ is associated with four (not necessarily distinct) numbers: $h_1(X) = CN(X^\infty)$, $h_m(X) = CN(X^\infty \cup R^\infty)$, $h_1^r(X) = CN(X^\infty)$ and $h_m^r(X) = CN(X^\infty \cup R^\infty)$. These definitions are naturally extended for sets of objects. Classifying objects by these numbers corresponds to four possible hierarchies, denoted $h_1, h_m, h_1^r$ and $h_m^r$ respectively. Jayanti showed that the first three hierarchies are not robust by showing that in some cases the consensus number of a combination of two objects (according to the appropriate definition) is greater than the consensus number of each of the objects. Consequently, the question whether $h_m^r$ is robust had become of special interest, and was studied in [BGA94, BNP94, CHJT94a, CHJT94b, PBN94].

Jayanti in [Jay95] states that if $h_m^r$ is robust, the difficult problem of computing the combined power of a set of objects reduces to the simpler problem of computing the power of the individual objects in the set. If, on the other hand, $h_m^r$ is not robust, it may be possible to
On the Robustness of $h^r_m$

(Preliminary Version, December 11, 1996)

Shlomo Moran$^{\dagger}$ and Lihu Rappoport$^{\ddagger}$
Faculty of Computer Science
The Technion, Israel

Abstract

We introduce an $N$-process deterministic concurrent object for $N \geq 3$ processes, called the conditional consensus object. This object, denoted as $\mathcal{W}$, is hard-wired in the sense that each process $P$ can access it using a single fixed port (though $P$ can use different ports in different copies of $\mathcal{W}$). We prove that $\mathcal{W}$ satisfies the following properties:

- There is no consensus protocol for three processes which uses many shared registers and many copies of $\mathcal{W}$ (and does not use any other object); but
- There is a consensus protocol for $N$ processes which uses one copy of $\mathcal{W}$ and $\binom{N}{3}$ copies of $\mathcal{CO}_3$, where $\mathcal{CO}_3$ is the standard consensus object for three processes.

This implies that the hierarchy $h^r_m$ is not robust for deterministic hard-wired objects.

---

$^{\dagger}$This work was supported by the France–Israel cooperative project: Graph Theoretical Methods in Distributed Computing 4474-2-93, and by the Bernard Elkin Chair in Computer Science. A preliminary version of this paper appears in the Proceedings of the 10th International Workshop on Distributed Algorithms, Bologna, Italy, October 1996, pp 344–361.

$^{\ddagger}$e-mail: moran@cs.technion.ac.il

$^{\dagger}$e-mail: lihu@cs.technion.ac.il. Part of this work was done while visiting at Labri, Université Bordeaux 1, France.