# A Highly Concurrent Algorithm for the Group Mutual Exclusion Problem

群馬大学工学部　高村政孝、五十嵐善英
(Masataka Takamura and Yoshihide Igarashi)

Department of Computer Science, Gunma University

**Abstract**

Group mutual exclusion is an interesting generalization of the mutual exclusion problem. This problem was introduced by Joung, and some algorithms for the problem have been proposed by incorporating mutual exclusion algorithms. Group mutual exclusion occurs naturally in a situation where a resource can be shared by processes of the same group, but not by processes of a different group. It is also called the congenial talking philosophers problem. In this paper we propose a highly concurrent algorithm based on ticket orders for the group mutual exclusion problem in the asynchronous shared memory model. It is a modification of the Bakery algorithm for the mutual exclusion problem. It uses single-writer shared variables together with two multi-writer shared variables that are never concurrently written. We show that the algorithm satisfies lockout freedom as well as group mutual exclusion.

## 1 Introduction

Mutual exclusion is a problem of managing access to a single indivisible resource that can only support one user at a time. The $k$-exclusion problem is a natural generalization of the mutual exclusion problem [2, 3, 12]. The group mutual exclusion problem is another natural extension of the mutual exclusion problem. This problem was introduced by Joung in [6], and some algorithms for the problem have been proposed [4, 6, 7, 8]. Group mutual exclusion is required in a situation where a resource can be shared by processes of the same group, but not by processes of a different group. A combination of $k$-exclusion and group mutual exclusion was also studied [14, 15]. The algorithms given in [4, 6, 8, 14, 15] use multi-writer/multi-reader shared variables in the asynchronous shared memory model.

As described in [6], group mutual exclusion can be described as the *congenial talking philosophers* problem. We assume that there are $n$ philosophers. They spend their time thinking alone. When a philosopher is tired of thinking, he/she attempts to attend a forum and to talk at the forum. We assume that there is only one meeting room. A philosopher wishing to attend a forum can do so if the meeting room is empty, or if some philosophers interested in the same forum as the philosopher in question are already in the meeting room. The *congenial talking philosophers* problem is to design an algorithm such that a philosopher wishing to attend a forum will eventually succeed in doing so. Philosophers interested in the same forum as the current forum held in the meeting room should be encouraged to attend it. This performance is measured as concurrency of attending a forum. In this paper, we propose an algorithms based on ticket orders for the group mutual exclusion problem in the asynchronous shared memory model.

## 2 Preliminaries

The computational model used in this paper is the asynchronous shared memory model. It is a collection of processes and shared variables. Processes take steps at arbitrary speeds, and there is no global clock. Interactions between a process and its corresponding philosopher are by input actions from the philosopher to the process and by output actions from the process to the philosopher. Each process is considered to be a state machine with arrows entering and leaving the process, representing its input and output actions. All communication among the processes is via shared memory.

A shared variable is said to be regular if every read operation returns either the last value written to the shared variable before the start of the read operation or a value written by one of the overlapping write operations [10]. A shared variable is said to be atomic if it is regular and the additional property that read operations and write operations behave as if they occur in some total order [10]. In this paper, we assume that all shared variables are atomic. The algorithms given in this paper uses single-writer/multi-reader shared variables, together with two multi-writer/multi-reader shared variables that are never concurrently written.

A philosopher with access to a forum is modeled as being in the talking region. When a philosopher is not involved in any forum, he/she is said to be in the thinking region. In order to gain admittance to the talking region, his/her corresponding process executes a trying protocol. The duration from the start of execution of the trying protocol to the entrance to the talking region is called the trying region. After the end of talking by a philosopher at a forum, his/her corresponding process executes an exit protocol. The duration of execution of the exit protocol is called the exit region. These regions are followed in cyclic order, from the thinking region to the trying region, to the talking region, to the exit region, and then back again to the thinking region. The congenial talking philosophers problem is to devise protocols for each philosopher to efficiently and fairly attend a forum when he/she wishes to talk under the conditions that there is only one meeting room and that only a single forum can be held in the meeting room at a time.

We assume $n$ philosophers, $P_1, P_2, ..., P_n$ who spend their time either thinking alone or talking in a forum. We also assume that there are $m$ different fora. Each philosopher $P_i$ $(1 \leq i \leq n)$ corresponds to process $i$. The inputs to process $i$ from philosopher $P_i$ are $try_i(f)$ which means a request by philosopher $P_i$ for access to the forum $f \in \{1, ..., m\}$ to talk there, and $exit_i$ which means an announcement of the end of talking by $P_i$. The outputs from process $i$ to philosopher $P_i$ are $talk_i$ which means granting attendance at the meeting room to $P_i$, and $think_i$ which means that $P_i$ can continue with his/her thinking alone without the use of the meeting room. These are external actions of the shared memory system. We assume that a philosopher in a forum spends an unpredictable but finite amount of time in the forum. The system to solve the congenial talking philosophers problem should satisfy the following conditions:

(1) **group mutual exclusion:** If some philosopher is in a forum, then no other philosopher can be in a different forum at the same time .

(2) **lockout freedom:** Any philosopher wishing to attend a forum eventually does so if each philosopher in any forum always leaves the forum.

(3) **progress for the exit region:** If a philosopher is in the exit region, then at some later point he/she enters the thinking region.

Waiting time and occupancy are important criteria to evaluate solutions to the congenial talking philosophers problem. Waiting time is the amount of time from when a philosopher wishes to attend a forum until he/she attends the forum. Concurrency is important to increase system performance concerning the resource. It is desirable for a solution to the congenial talking philosophers problem to satisfy the following property called *concurrent occupancy* [4, 7, 8].

(4) **concurrent occupancy:** If some philosopher $P$ requests to attend a forum and no philosopher is currently attending or requesting a different forum, then $P$ can smoothly attend the forum without waiting for other philosophers to leave the forum.

## 3 A Highly Concurrent Algorithm

The following procedure $(n, m)$-*HCGME* is a modification of the Bakery algorithm for the mutual exclusion problem [2, 9, 11], where $n$ is the number of philosophers, $m$ is the number of different fora, and $N$ is the set of natural numbers. A capturing technique is used to improve the concurrency performance of the algorithms for the group mutual exclusion problem in [6, 7, 8]. The algorithm proposed in this paper uses a door in the doorway. A chair is chosen in each current forum. When the chair wishes to leave the forum, he/she closes the door to prevent other philosophers from entering the same forum. While the door is open, philosophers wishing to enter the same forum as the current forum held in the meeting room are allowed to enter it. In this way, concurrency can be improved. All shared variables, except for *door* and *chair*, are single-writer/multi-reader shared variables. Both *door* and *chair* are multi-writer/multi-reader shared variables, but no concurrency of writing operations occurs concerning either of these two shared variables.

```
procedure $(n,m)$-HCGME
shared variables
  for every $i \in \{1,...,n\}$:
    $transit(i) \in \{0,1\}$, initially 0, writable by process $i$ and readable by
        all processes $j \neq i$;
    $checkdw(i) \in \{0,1\}$, initially 0, writable by process $i$ and readable
        by all processes $j \neq i$;
    $ticket(i) \in N$, initially 0, writable by process $i$ and readable by
        all processes $j \neq i$;
    $forum(i) \in \{0,1,...,m\}$, initially 0, writable by process $i$ and
        readable by all processes $j \neq i$;
  $forum(0)$, always 0, readable by all processes;
  $door \in \{open, close\}$, initially $open$, writable and readable by
```

```
    all processes (but never concurrently written);
  $chair \in \{0, 1, ..., n\}$, initially 0, writable and readable by all processes
    (but never concurrently written);

process $i$
  input actions {inputs to process $i$ from philosopher $P_i$}:
    $try_i(f)$ for every $1 \leq f \leq m$, $exit_i$;
  output actions {outputs from process $i$ to philosopher $P_i$}:
    $talking_i$, $thinking_i$;

  ** thinking region **

01: $try_i(f)$:
02: $transit(i) := 1$;
03: if $door = close$ then begin
04:    $checkdw(i) := 1$;
05:    waitfor $door = open$;
06:    $checkdw(i) := 0$ end;
07: $ticket(i) := 1 + max_{j \neq i} ticket(j)$;
08: $forum(i) := f$;
09: $transit(i) := 0$;
10: for each $j \neq i$ do begin
11:    waitfor $transit(j) = 0$ or $checkdw(j) = 1$ or $f = forum(chair)$;
12:    waitfor $ticket(j) = 0$ or $(ticket(i), f, i) < (ticket(j), forum(j), j)$
         or $f = forum(chair)$ end;
13: if for each $j \neq i$, $ticket(j) = 0$ or $(ticket(i), f, i) < (ticket(j), forum(j), j)$
       then $chair := i$;
14: $talking_i$;

  ** talking region **

15: $exit_i$;
16: if $chair = i$ then begin
17:    $door := close$;
18:    for each $j \neq i$ do begin
19:       waitfor $transit(j) = 0$ or $checkdw(j) = 1$;
20:       waitfor $f \neq forum(j)$ or $ticket(j) = 0$ end;
21:    $chair := 0$;
22:    $door := open$;
23:    for each $j \neq i$ do
24:       waitfor $checkdw(j) = 0$ end;
25: $forum(i) := 0$;
26: $ticket(i) := 0$ ;
27: $thinking_i$;
```

Relation $(a, b, c) < (a', b', c')$ in the procedure is lexicographical order. The process with the smallest triple of nonzero ticket number, forum, and its identifier is chosen as the chair of the forum held in the meeting room. If a philosopher $P_i$ in a forum is not the chair, $P_i$ can smoothly leave the talking region by resetting $ticket(i)$ and $forum(i)$. However, when the chair wishes to leave the forum, he/she must close the door of the doorway and be waiting for all philosophers in the same forum to leave. After the chair observes that all philosophers in the forum have left, he/she resigns the chair and opens the door in the doorway. Then after confirming that all philosophers waiting at line 05 have noticed that the door has opened, the philosopher who resigned the chair leaves the forum. The proof of the next theorem is omitted here due to the page limit.

**Theorem 1** *In any execution by $(n, m)$-HCGME, shared variable chair is never concurrently written, and once chair is set to be $i$ $(1 \leq i \leq n)$, the contents of chair remains $i$ until it is reset to be 0 by process $i$ at line 21 of the program. In any execution by $(n, m)$-HCGME, shared variable door is also never concurrently written.*

**Theorem 2** *$(n, m)$-HCGME guarantees group mutual exclusion, lockout freedom, and progress for the exit region.*

*Proof.* The current chair is uniquely determined by *chair*. While a philosopher is the chair, only philosophers wishing to attend the same forum as the chair's forum are allowed to enter the forum. When the chair wishes to leave the forum, he/she closes *door* in the exit region to prevent a newcomer to the trying region from getting a ticket. After observing that all other philosophers in the forum have left, the chair philosopher resigns the chair and opens *door*. Hence, group mutual exclusion is guaranteed. The time from when a philosopher enters the trying region until he/she enters the talking region is bounded by $2tc + O(nt)l$, where $t = min\{n, m\}$, $c$ is an upper bound on the time that any philosopher spends in the talking region, and $l$ is an upper bound on the time between two successive atomic steps by a process. Hence, lockout freedom is guaranteed. Progress for the exit region is obvious. $\square$

We can reduce the waiting time for processes wishing to attend the forum of the chair by modifying $(n, m)$-HCGME. For this purpose, we extend the domain of *shared* variable *door* to $\{close, 0, 1, ..., m\}$. If the forum held by the chair is stored into *door*, then at the beginning of the trying region each philosopher checks whether his/her forum of interest is the same as the forum indicated in *door*. If a philosopher notices that he/she wishes to attend the same forum as the forum shown in *door*, the philosopher sets his/her ticket number as well as forum number to be the same as the ticket number and forum number of the chair, and then he/she is granted attendance to the forum. In this way, such philosophers can attend the forum in O(1) atomic steps after the entrance to the trying region. This technique is called *smooth admission*.

## 4 Concluding Remarks

The algorithm given in this paper, $(n, m)$-HCGME, uses single-writer shared variables together with two multi-writer shared variables that are never concurrently written. The concurrency performance of our algorithm is superior to the algorithms in [4, 8]. The ticket domain of our algorithm is unbounded as in the Bakery algorithm. At present we do not know whether we can modify our algorithm using similar techniques given in [1, 5, 13] so that the ticket domain is bounded. This problem is worthy of further investigation.

## References

[1] U. Abraham, "Bakery algorithms", Technical Report, Dept. of Mathematics, Ben Gurion University, Beer-Sheva, Israel, 2001.

[2] H. Attiya and J. Welch, " Distributed Computing: Fundamentals, Simulations and Advanced Topics", McGraw-Hill, New York, 1998.

[3] M. J. Fischer, N. A. Lynch, J. E. Burns, and A. Borodi, "Resource allocation with immunity to limited process failure", *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico: 234–254, 1979.

[4] V. Hadzilacos, "A note on group mutual exclusion", *Proceedings of 12th Annual ACM Symposium on Principles of Distributed Computing*, Newport, Rhode Island, pp.100–106, 2001.

[5] P. Jayanti, K. Tan, G. Friedland, and A. Katz, "Bounded Lamport's bakery algorithm", *Proceedings of SOFSEM'2001, Lecture Notes in Computer Science*, vol.2234, Springer-Verlag, Berlin, pp.261–270, 2001.

[6] Yuh-Jzer Joung, "Asynchronous group mutual exclusion", *Distributed Computing*, vol.13, pp.189–206, 2000.

[7] Yuh-Jzer Joung, "The congenial talking philosophers problem in computer networks", *Distributed Computing*, vol.15, pp.155–175, 2002.

[8] P. Keane and M. Moir, "A simple local-spin group mutual exclusion algorithm", *IEEE Transactions on Parallel and Distributed Systems*, vol.12, 2001.

[9] L. Lamport, "A new solution of Dijkstra's concurrent programming problem", *Communications of the ACM*, vol.17, pp.453–455, 1974.

[10] L. Lamport, "The mutual exclusion problem. Part II : Statement and solutions", *J. of the ACM*, vol. 33, pp.327–348, 1986.

[11] N. A. Lynch, "Distributed Algorithms", Morgan Kaufmann, San Francisco, California, 1996.

[12] M. Omori, K. Obokata, K. Motegi and Y. Igarashi, "Analysis of some lockout avoidance algorithms for $k$-exclusion problem", *Interdisciplinary Information Sciences*, vol.8, pp.187–198, 2002.

[13] M. Takamura and Y. Igarashi, "Simple mutual exclusion algorithms based on bounded tickets on the asynchronous shared memory model", *IEICE Transactions on Information and Systems*, vol.E86-D, pp.246–254, 2003.

[14] K. Vidyasankar, "A highly concurrent group mutual $l$-exclusion algorithm", *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, Monterey, California, p.130, 2002.

[15] K. Vidyasankar, "A simple group mutual $l$-exclusion algorithm", to appear in *Information Processing Letters*.