Asynchronous Group Mutual Exclusion*

Yuh-Jzer Joung†
joung@ccms.ntu.edu.tw
Department of Information Management
National Taiwan University
Taipei, Taiwan

Abstract

Mutual exclusion and concurrency are two fundamental and essentially opposite features in distributed systems. However, in some applications such as Computer Supported Cooperative Work (CSCW) we have found it necessary to impose mutual exclusion on different groups of processes in accessing a resource, while allowing processes of the same group to share the resource. To our knowledge, no such design issue has been previously raised in the literature.

In this paper we address this issue by presenting a new problem, called *Congenial Talking Philosophers*, to model group mutual exclusion. We also propose several criteria to evaluate solutions of the problem and to measure their performance. Finally, we provide an efficient and highly concurrent distributed algorithm for the problem in a shared-memory model where processes communicate by reading from and writing to shared variables. The distributed algorithm meets the proposed criteria, and has performance similar to some naive but centralized solutions to the problem.

*An extended abstract of the paper appeared in *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, pp. 51-60 (Puerto Vallarta, Mexico, June 28–July 2, 1998). This research was supported in part by the National Science Council, Taipei, Taiwan, under Grants NSC 85-2213-E-002-059, NSC 86-2213-E-002-053, and NSC 87-2218-E-002-050, and by the 1997 Research Award of College of Management, National Taiwan University.

†The author is currently visiting Laboratory for Computer Science, Massachusetts Institute of Technology (1999-2000).

1 Introduction

Mutual exclusion and concurrency are two fundamental issues in distributed systems. Mutual exclusion guarantees exclusive access to a common resource to one of a set of competing processes, while concurrency allows processes to share a resource to increase system performance. In spite of their contradictory nature, in some applications such as Computer Supported Cooperative Work (CSCW) we have found it necessary to guarantee mutual exclusion while still exploiting a certain degree of concurrency.

For example, consider a video-conferencing system with an electronic white board. A user can use this white board to post information that she/he would like to share with others. All the information posted on the white board will be seen by all users currently online. Thus, when a group of users uses the system to discuss some issue, another group of users whose interests conflict with the first group must be excluded from using the system. On the other hand, when some user is using in the system, we wish to encourage discussion by allowing more users with the same interest to use the system. Thus, a design which involves both mutual exclusion and concurrency is required.

As another example, consider several users working on a project that has some large data objects stored on a secondary memory device (such as a CD jukebox). When a user needs to access a data object, the data object is loaded from the device to a cache buffer. To increase performance, once a data object is loaded it will remain in the buffer until another data object is requested. So while a data object resides in the buffer, users that need to work on this data object are allowed to access the buffer concurrently, and users that need a different data object have to wait until no user is working on the data object currently in the buffer. That is, users with the same interests can concurrently access the buffer, while users with different interests must be excluded from accessing the buffer.

Although many systems may require that processes of the same group share a resource while processes of different groups use the resource exclusively, to our knowledge, group mutual exclusion has not been previously raised in the literature. Note that we do not require processes of the same group to synchronize in order to access the resource. Problems concerning *synchronous* group mutual exclusion, where a set of processes must synchronize in order to access a resource or a process must possess all needed resources in order to continue, have been addressed by Chandy and Misra [7, 8].

In this paper we present a problem, called *Congenial Talking Philosophers*, to model group mutual exclusion. The problem concerns a set of n philosophers which spend their time thinking alone and talking in fora. Given that there is only one meeting room (the critical section), a philosopher attempting to attend a forum can succeed only if the meeting room is empty (and in this case the philosopher starts the forum), or some philosopher interested in the same forum is already in the meeting room (and in this case the philosopher joins the ongoing forum). The challenge is to design an algorithm for the philosophers to ensure that a philosopher attempting to attend a forum will eventually succeed, while at the same time encouraging philosophers interested in the same forum to be in the meeting room simultaneously. In this paper we focus on solutions in the shared-memory model, where processes communicate by reading from and writing to shared variables. Solutions based on message passing are considered in a separate paper [16].

The Congenial Talking Philosophers problem is related to some fundamental problems in distributed systems. For example, by dedicating one forum to each philosopher, the problem is reduced to n-process mutual exclusion where only one process can be in the critical section at a time. The problem can also be reduced to the Readers and Writers problem [9] where a shared object can be concurrently read by different processes, while writing alone must be mutually exclusive. To do so, we can employ a READ operation (forum) for all processes (philosophers) in the system, and a unique WRITE operation for each individual one. A process attempting to read the shared object then requests the READ operation to access the object, while it requests its own WRITE operation when it wishes to update the object. Thus the Congenial Talking Philosophers problem is more general than the two classical problems.

Note that resolving conflicts between READ/WRITE and WRITE/WRITE operations while facilitating concurrency among READ operations is the central topic of database concurrency control (see, e.g., [11, 21, 19, 18, 5, 26, 2]). Despite the similar objective, the Congenial Talking Philosophers problem targets the construction of a low-level mechanism to support operation execution. In contrast, database concurrency control typically uses such mechanisms (e.g., locking) to ensure serializability at the transaction level.

Intuitively, a maximal degree of concurrency can be achieved if philosophers are allowed to attend a forum while some philosopher with the same interest is occupying the meeting room. However, given that each philosopher independently determines when it will be interested in a forum and how long it will stay in a forum (although it can only spend a finite amount of time in the forum), such a degree of concurrency cannot be achieved if we are also to ensure a bounded delay on the time a philosopher spends in waiting for a forum. This is because otherwise two philosophers interested in the same forum may repeatedly enter the meeting room, thus blocking a third philosopher waiting for a different forum indefinitely. So the challenge of the problem lies in the exploitation of a high degree of concurrency in attending a forum while ensuring a minimum delay for the philosophers waiting for a different forum.

Indeed, the problem is much more difficult than we originally had thought. Figure 4 in Section 5 gives the final version of our main algorithm, which consists of only 13 lines of statements. In the process of designing the algorithm, we made several mistakes, some of which were quite subtle and occurred only in the presence of concurrency. We also discovered several performance trade-offs—concurrency vs. waiting time—by simply reversing the execution order of two statements which looks irrelevant at first glance! We shall present some of the findings in our discussion of the algorithm.

The rest of the paper is organized as follows. Section 2 presents the Congenial Talking Philosophers problem in more detail, and proposes criteria that can be used to evaluate solutions of the problem and to measure their performance. For comparison, we first offer some simple but centralized solutions in Section 3. Section 4 then presents a fully distributed solution where philosophers may only attend two fora, and Section 5 generalizes the solution to an arbitrary number of fora. Section 6 discusses related work and concludes.

2 The Congenial Talking Philosophers Problem

We consider a set of n philosophers $p_0, p_1, \ldots, p_{n-1}$ which spend their time either thinking alone or talking in fora. The philosophers may like to hold m different fora $\mathsf{F}_0, \mathsf{F}_1, \ldots, \mathsf{F}_{m-1}$ but, due to the capacity of the meeting room, only one forum can be held at a time. However, more than one philosopher can be in a forum simultaneously. Initially, all the philosophers are thinking. When a philosopher is tired of thinking, it chooses a forum to attend. We assume that when a philosopher attends a forum, it spends an unpredictable but finite amount of time in the forum. After a philosopher leaves a forum, it returns to thinking.[1] We say that a forum is *in session* if some philosopher is in the forum. The *Congenial Talking Philosophers* problem consists of the following

[1]Throughout the paper, "in a forum" is used synonymously with "in the meeting room." Likewise, "to attend/leave a forum" is synonymous with "to enter/exit the meeting room."

requirements:[2]

mutual exclusion: if some philosopher is in a forum, then no other philosopher can be in a different forum at the same time.

bounded delay: a philosopher attempting to attend a forum will eventually succeed.

We are seeking solutions that facilitate **concurrent entering**, meaning that if some philosophers are interested in a forum and no philosopher is interested in a different forum, then the philosophers can concurrently enter the meeting room to hold the forum. As discussed in Section 1, the n-process mutual exclusion problem is a special case of Congenial Talking Philosophers in which only one philosopher may attend each forum. Obviously, it would be overkill to solve Congenial Talking Philosophers using solutions for n-process mutual exclusion (e.g., [10, 6, 22]). So a reasonable solution for the problem must allow philosophers to share the meeting room when no one is interested in a different forum.

The concurrent entering requirement we have defined above is slightly stronger as it requires philosophers not just to be able to be *in* the meeting room simultaneously, but to *enter* the meeting room concurrently. The intent of this stronger condition is to prevent unnecessary synchronization among philosophers attending a forum when no one else is interested in a different forum. Such synchronization occurs, for example, in solutions that use a shared variable to control the use of the meeting room, and philosophers access the shared variable in a mutually exclusive style to avoid conflicts. Such solutions allow philosophers to be in the meeting room simultaneously, but do not allow them to enter the meeting room concurrently because of the synchronization imposed on the philosophers in accessing the shared variable. The overhead of such solutions is especially high when the number of fora the philosophers would like to hold is relatively small compared to the number of philosophers that are interested in each forum.

Solutions of Congenial Talking Philosophers can be evaluated by two parameters: *time* and *concurrency*. For the time parameter, we are concerned with how long a philosopher may wait before entering a forum. Instead of using physical time—which would be system dependent and hard to analyze, we use *passages* as the basic metric for evaluating time, as defined below:

[2] We assume *finite progress* for the philosophers, meaning that if a philosopher is given an instruction then it will execute the instruction in finite but unpredictable time. Moreover, we assume that basic machine-level instructions such as *read* and *write* to a shared variable are executed atomically.

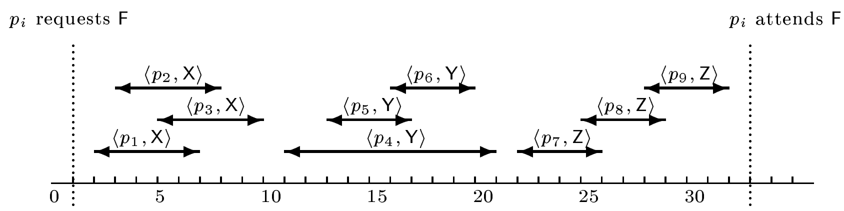

Figure 1: A layout of passages.

***Definition* 2.1** *A **passage** by p_i through a forum F is an interval $[t_1, t_2]$, where t_1 is the time p_i enters the forum, and t_2 the time p_i leaves the forum. The passage is **initiated** at t_1, and **completed** at t_2. The passage is **ongoing** at any time in between t_1 and t_2.*

A passage by p_i through a forum F is represented by $\langle p_i, \mathsf{F}\rangle$, which we refer to as the ***attribute*** of the passage. When no confusion is possible, we use intervals and attributes interchangeably to represent passages (where intervals are denoted by square brackets $[t_1, t_2]$, and attributes by angle brackets $\langle p_i, \mathsf{F}\rangle$).

Due to concurrency, passages may overlap. Therefore, an explicit count of the total number of passages a philosopher may wait is not a good metric for the waiting time. Instead, we measure the waiting time by counting the minimal number of passages that are sufficient to "cover" all the passages in consideration. For example, suppose that a philosopher p_i, after requesting a forum F, waits for the nine passages shown in Figure 1 before it can attend F. Then, since $\langle p_2, \mathsf{X}\rangle$ is covered by $\langle p_1, \mathsf{X}\rangle$ and $\langle p_3, \mathsf{X}\rangle$, and since $\langle p_5, \mathsf{Y}\rangle$ and $\langle p_6, \mathsf{Y}\rangle$ are covered by $\langle p_4, \mathsf{Y}\rangle$, only the six passages $\langle p_1, \mathsf{X}\rangle$, $\langle p_3, \mathsf{X}\rangle$, $\langle p_4, \mathsf{Y}\rangle$, $\langle p_7, \mathsf{Z}\rangle$, $\langle p_8, \mathsf{Z}\rangle$, and $\langle p_9, \mathsf{Z}\rangle$ account for the delay in satisfying p_i's request.

A formal definition is given below.

***Definition* 2.2** *Let $\mathcal{S}$ be a set of intervals. A subset $\mathcal{R}$ of $\mathcal{S}$ is a **cover** of $\mathcal{S}$ if for every $\alpha \in \mathcal{S}$, every time instance in α is in some $\beta \in \mathcal{R}$ (that is, $\forall [t_1, t_2] \in \mathcal{S} : t_1 \leq t \leq t_2 \Rightarrow \exists [t_3, t_4] \in \mathcal{R}, t_3 \leq t \leq t_4$). It is **minimal** if for any other cover $\mathcal{R}'$ of $\mathcal{S}$, $|\mathcal{R}| \leq |\mathcal{R}'|$. The **dimension** of $\mathcal{S}$, denoted by $\mathbf{dim}(\mathcal{S})$, is the size of a minimal cover of $\mathcal{S}$.*

To illustrate the definition, the set of passages shown in Figure 1 has a minimal cover consisting of $\langle p_1, \mathsf{X}\rangle$, $\langle p_3, \mathsf{X}\rangle$, $\langle p_4, \mathsf{Y}\rangle$, $\langle p_7, \mathsf{Z}\rangle$, $\langle p_8, \mathsf{Z}\rangle$, and $\langle p_9, \mathsf{Z}\rangle$, and so has a dimension 6.

The ***time complexity*** of an algorithm for the Congenial Talking Philosophers problem is measured by the maximal dimension of the set of passages a philosopher has to wait before it can attend a forum.

Note that, still, the dimension does not necessarily truly reflect the elapsed time. This is because in real applications consecutive passages through the critical section by different groups of processes usually require "context switches". For example, when a user requests a different data object in the CSCW environment described in Section 1, the storage device has to unload the old object and then load the new one. Since such loading and unloading are usually time-consuming, in the Congenial Talking Philosophers problem a philosopher waiting for more passages through the same forum may in practice need less time than one waiting for fewer passages through different fora. The notion of "rounds of passages" is therefore conceived to measure the number of "context switches" (i.e., forum switches).

Definition* 2.3** *Let $\mathcal{S}$ be a set of passages through forum* F*. Let $t_s = \min\left\{t \,\middle|\, [t,t'] \in \mathcal{S}\right\}$, and $t_f = \max\left\{t' \,\middle|\, [t,t'] \in \mathcal{S}\right\}$. Then, $\mathcal{S}$ is a* ***round of passages through F *(or simply* ***a round of*** F*) if the following two conditions are satisfied:*

1. *No passage other than those in $\mathcal{S}$ is initiated in between t_s and t_f.*
2. *The last passage initiated before t_s and the first passage initiated after t_f, if they exist, are for a forum other than* F*.*

If $\mathcal{S}$ is a round of F*, then we say that it* ***starts*** *at t_s, and* ***terminates*** *at t_f. It is* ***ongoing*** *in between t_s and t_f.*

In other words, a round of F is a maximal set of consecutive passages through F. For example, the three passages $\langle p_4, \mathsf{Y}\rangle$, $\langle p_5, \mathsf{Y}\rangle$, and $\langle p_6, \mathsf{Y}\rangle$ in Figure 1 constitute a round of Y, and the three passages $\langle p_7, \mathsf{Z}\rangle$, $\langle p_8, \mathsf{Z}\rangle$, and $\langle p_9, \mathsf{Z}\rangle$ constitute a round of Z.

The ***forum-switch complexity*** is measured by the maximum number of rounds of passages a philosopher may wait before it can attend a forum.

For example, suppose that in the worst case p_i has to wait for the following sequence of passages before it can attend a forum:

$$\langle p_0, \mathsf{F}_0\rangle, \langle p_1, \mathsf{F}_1\rangle, \langle p_2, \mathsf{F}_0\rangle, \ldots, \langle p_{2k-1}, \mathsf{F}_1\rangle$$

Then, both the time complexity and the forum-switch complexity are $2k$. On the other hand, if the sequence is

$$\langle p_0, \mathsf{F}_0\rangle, \langle p_2, \mathsf{F}_0\rangle, \ldots, \langle p_{2k-2}, \mathsf{F}_0\rangle, \langle p_1, \mathsf{F}_1\rangle, \langle p_3, \mathsf{F}_1\rangle, \ldots, \langle p_{2k-1}, \mathsf{F}_1\rangle$$

where the passages through the same forum do not overlap, then the time complexity is still $2k$, but the forum-switch complexity is only 2. If the average length of a passage is t and the time to perform a context switch is s, then the total time p_i has to wait in the first case above is $2k(t+s)$, and in the second case is $2kt+2s$.

The ***degree of concurrency*** is defined by the maximum number of passages that can be initiated while some philosopher is in the meeting room and another philosopher is waiting for a different forum. Intuitively, because of mutual exclusion, when a philosopher p is in the meeting room no other philosopher interested in a different forum can use the meeting room. Given that a philosopher p decides on its own how long it will use the meeting room, better resource utilization can be achieved if we allow more philosophers interested in the same forum to share the meeting room with p. So a higher degree of concurrency implies better resource utilization.

Concurrency may also be measured in other ways, for example, by the maximum number of philosophers that can be in the meeting room simultaneously. However, if the problem definition allows k philosophers to hold a forum F, then most solutions that facilitate concurrent entering would probably allow, in the best case, k philosophers to be in F simultaneously. So this is not a useful metric for concurrency.

Because we do not assume any lower or upper bounds on the time a philosopher spends in a meeting room (except that the time is finite), it is possible for an algorithm to achieve a virtually "unbounded" degree of concurrency. For example, if an algorithm is such that while some p_i is in the meeting room any other p_j interested in the same forum can enter the meeting room, then since there is no limit on how quickly p_j will finish the forum and re-request the forum, p_j may enter/re-enter the meeting room any number of times. Although this number is finite, it is not bounded by any predetermined constant. Note that this does not contradict bounded delay as p_i will eventually leave the meeting room. For a more detailed example, see the following section.

3 Some Simple Centralized Solutions

A simple solution can be obtained by employing a concierge to schedule fora. The concierge periodically inspects philosophers' states to see if anyone is interested in a forum, and then adopts some scheduling policy to guarantee mutual exclusion, bounded delay, and concurrent entering[3]. For example, if no forum is

[3]One could argue that the algorithm does not facilitate concurrent entering because the concierge has to observe philosophers' states in some sequential order. So when philosophers are ready for the same forum, their entries to the forum cannot be truly concurrent as the

currently in session, the concierge can schedule the first waiting philosopher it has seen to start a forum. All philosophers that are also ready for the same forum also start the forum simultaneously, and philosophers that are interested in different fora are queued. While a forum is in session, the concierge can choose a fixed philosopher as a reference so that while the reference philosopher stays in the forum, every other philosopher attempting to attend the forum can do so. Once the reference philosopher leaves the forum, if some other philosopher is waiting for a different forum, then the "door" to the forum is closed and no other philosopher can join the forum; otherwise, a new reference philosopher is chosen. Note that bounded delay can be guaranteed because the reference philosopher can only spend a finite amount of time in a forum. The complete algorithm, which we refer to as CTP-C (an abbreviation for Congenial Talking Philosophers-Centralized), can be found in [15].

It is easy to see that CTP-C offers an unbounded degree of concurrency. This is because while a reference philosopher is in a forum, another philosopher can repeatedly attend the forum, leave the forum, and become interested again in the forum. The analysis of the algorithm's forum-switch complexity and time complexity is somewhat tedious and details have been provided in [15]. For comparison with our distributed algorithm, we summarize the result here: after a philosopher p_i requests F_k, it waits for at most $m+1$ rounds of passages before a round of F_k is initiated in which it can make a passage through F_k, where m is the number of fora the philosophers may like to hold. For each such round, the dimension of the passages in the round is $O(n)$, where n is the total number of philosophers. That is, CTP-C has forum-switch complexity $m + 1$, and time complexity $O(n \cdot m)$.

The algorithm can be made more distributed by employing a concierge for each forum. To ensure mutual exclusion, a token is shared by the concierges. A concierge must possess the token before scheduling any philosopher to attend a forum, and it must keep the token until all philosophers have left the forum. To increase concurrency, a concierge may allow a philosopher to re-attend the forum while it possesses the token. To also guarantee bounded delay, a reference philosopher can be chosen similarly to CTP-C to control forum admission. Competition for the token among the concierges can be solved by calling an m-process mutual exclusion algorithm. The complete code of the new algorithm, which we refer to as CTP-SD (SD for Semi-Distributed), can also be found in [15].

order depends on how the concierge observes their readiness. The situation is similar in the following algorithm where a concierge is employed for each forum.

Like CTP-C, the use of reference philosophers also allows CTP-SD to claim an unbounded degree of concurrency. Unlike CTP-C, however, the forum-switch complexity and time complexity depend on the fairness notion guaranteed by the underlying m-process mutual exclusion algorithm, and, in the worst case, both could be unbounded! To see this, suppose that a concierge C that manages a forum F does not request the token until it has observed a request for F (so that competition for the token is only among the concierges that need it). Then, after a philosopher has requested F, depending on C's execution speed, other concierges may alternatively schedule an arbitrary finite number of passages through different fora before C requests the token. It then follows that both the forum-switch complexity and the time complexity are unbounded. On the other hand, the problem may be overcome by letting the token circulate among all concierges, regardless of whether or not they have received a forum request. Clearly, this approach is not adequate if concierges' loads are not balanced, i.e., some fora are requested more often than others.

Still, CTP-SD is not fully distributed because the scheduling of entries to the same forum is operated by a single concierge. As we have seen, a slow concierge may cause poor time and forum-switch performance. Moreover, when the number of possible fora m is greater than the number of philosophers n, then we will need more active processes as opposed to CTP-C, and when m is small, the concierges become a bottleneck as in CTP-C.

In the following sections we present a fully distributed solution for the Congenial Talking Philosophers problem. The algorithm has similar time complexity and forum-switch complexity to CTP-C, and achieves a degree of concurrency of $O(n^3)$.

4 A Distributed Solution for Two Fora

We first present a distributed solution for the Congenial Talking Philosophers problem with only two fora F and $\overline{\mathsf{F}}$. To help understand our solution, we present it in stages.

4.1 A First Attempt

Consider the algorithm presented in Figure 2, which we refer to as CTP-Temp. CTP-Temp uses the following variables:

- *turn* : $(\mathsf{F}, \overline{\mathsf{F}})$; a variable shared (with read/write access) by the philosophers to identify the forum that currently has priority to start. It is initialized arbitrarily.

- *flag* : **array** $[0..n-1]$ of $\langle state, op\rangle$, where $flag[i]$ records p_i's state and the forum it wishes to attend. There are three possible states: *passive*, *request*, and *in_cs*. State *passive* means that the philosopher does not intend to attend any forum; *request* means that the philosopher wishes to attend some forum; and *in_cs* means that the philosopher has obtained a "temporary" permission for its request. A philosopher in state *in_cs* may be prevented from attending a forum if some other philosopher has also obtained a temporary permission for its request. *op* ranges over $\mathsf{F}, \overline{\mathsf{F}}$, and $\perp$, where $\perp$ means that no forum is requested by p_i.

  $flag[i]$ can be read/written by p_i, but other philosophers can only read it.

  Note that although the value of each $flag[i]$ is represented as a pair, we require access to $flag[i]$ to be atomic. This can be done using a straightforward encoding as each pair $\langle state, op\rangle$ can range over only 3×3 possible values. Denoting $flag[i]$ as a pair helps us understand its meaning.

The predicate $all\_passive(\overline{\mathsf{F}})$ defined in Figure 2 checks if no philosopher is interested in $\overline{\mathsf{F}}$, and $none\_in\_cs(\overline{\mathsf{F}})$ determines if no philosopher has obtained a temporary permission to attend $\overline{\mathsf{F}}$.

As can be seen, CTP-Temp bears some similarity to Knuth's 2-process mutual exclusion algorithm [17]. It employs a variable *turn* to resolve the conflict between philosophers attempting different fora. When a philosopher p_i wishes to attend a forum, say F, it checks if F has priority (i.e., $turn = \mathsf{F}$), or no one is interested in $\overline{\mathsf{F}}$ (i.e., $all\_passive(\overline{\mathsf{F}}) = true$). It cannot proceed until one of the two conditions holds. Then, p_i obtains a temporary permission to attend F (line 4). To actually attend F, p_i must further check if any philosopher has obtained a temporary permission to attend $\overline{\mathsf{F}}$ (by evaluating $none\_in\_cs(\overline{\mathsf{F}})$). If so, p_i gives up its temporary permission and loops back to check if F still has priority over $\overline{\mathsf{F}}$ or no one is interested in $\overline{\mathsf{F}}$. If this time no philosopher obtains a temporary permission to attend $\overline{\mathsf{F}}$ then p_i can start F; otherwise p_i must repeat the loop. After finishing the forum, p_i switches *turn* to $\overline{\mathsf{F}}$ so that philosophers interested in $\overline{\mathsf{F}}$ will then have priority to proceed.

CTP-Temp inherits the mutual exclusion property from Knuth's algorithm. To see this, observe that a philosopher p_i attends F only if $none\_in\_cs(\overline{\mathsf{F}})$ holds. Since p_i tests $none\_in\_cs(\overline{\mathsf{F}})$ only after it has set $flag[i]$ to $\langle in\_cs, \mathsf{F}\rangle$, it is never the case that another philosopher attempting $\overline{\mathsf{F}}$ evaluates a true value for $none\_in\_cs(\mathsf{F})$ simultaneously. Moreover, when more than one philosopher is in F, the switch of *turn* in line 7 by any one of them cannot incur a pending $\overline{\mathsf{F}}$

/* assuming p_i is attempting F */
1 **repeat**
2 $\quad$ $\mathit{flag}[i] \leftarrow \langle \mathit{request}, \mathsf{F} \rangle$;
3 $\quad$ **while** $\mathit{turn} \neq \mathsf{F} \wedge \neg \mathit{all\_passive}(\overline{\mathsf{F}})$ **do** **skip** ;
4 $\quad$ $\mathit{flag}[i] \leftarrow \langle \mathit{in\_cs}, \mathsf{F} \rangle$;
5 **until** $\mathit{none\_in\_cs}(\overline{\mathsf{F}})$
6 $<<$ talk in F $>>$
7 $\mathit{turn} \leftarrow \overline{\mathsf{F}}$;
8 $\mathit{flag}[i] \leftarrow \langle \mathit{passive}, \perp \rangle$;

where

$\mathit{all\_passive}(\overline{\mathsf{F}}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : \mathit{flag}[j] = \langle \mathit{state}, \mathit{op} \rangle \Rightarrow \mathit{op} \neq \overline{\mathsf{F}}$

$\mathit{none\_in\_cs}(\overline{\mathsf{F}}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : \mathit{flag}[j] = \langle \mathit{state}, \mathit{op} \rangle \Rightarrow (\mathit{state} \neq \mathit{in\_cs} \vee \mathit{op} = \mathsf{F})$

Figure 2: Algorithm CTP-Temp.

because so long as some philosopher is in F its flag remains $\langle \mathit{in\_cs}, \mathsf{F} \rangle$, and thus $\mathit{none\_in\_cs}(\mathsf{F})$ evaluates to *false*. So no philosopher attempting $\overline{\mathsf{F}}$ can skip the loop condition in line 5 to attend $\overline{\mathsf{F}}$.

CTP-Temp also facilitates concurrent entering because if no philosopher has requested $\overline{\mathsf{F}}$, then $\mathit{all\_passive}(\overline{\mathsf{F}})$ evaluates to *true*. So all philosophers attempting to attend F can do so, and their entries to the forum are mutually independent.

In the presence of concurrency, unfortunately, CTP-Temp fails to guarantee bounded delay. To illustrate, assume that p_i repeatedly wishes to attend F, while p_j repeatedly wishes to attend $\overline{\mathsf{F}}$. A slow philosopher p_k attempting to attend, say F, may be forever blocked in the while-loop of the algorithm if every time when p_k checks the loop condition it sees that $\mathit{turn} = \overline{\mathsf{F}}$ and $\mathit{all\_passive}(\overline{\mathsf{F}}) = \mathit{false}$; and within the interval of two consecutive checks, p_j and p_i each have completed an instance of $\overline{\mathsf{F}}$ and F, respectively.

4.2 A Fair Solution

The unbounded delay of CTP-Temp is due to the fact that when $\mathit{turn} = \mathsf{F}$, some philosopher attempting to attend F will succeed, but we cannot guarantee which philosopher will succeed. To overcome this problem, we let some philosopher attending F "capture" all philosophers currently waiting for the same forum into the forum.[4] Obviously, not every philosopher can capture philosophers, for otherwise philosophers interested in F will keep entering F, thus blocking the other forum indefinitely. Our approach is to let the first philosopher starting a new session of F act as a *captain* to capture philosophers. The philosophers that are captured into F are called the *successors* of the captain. Successors are not allowed to capture philosophers to avoid possibility of livelock.

The following variable is added to assist the coordination:

- *successor* : **array** $[0..n-1]$ of $(\mathsf{F}, \overline{\mathsf{F}}, \perp)$, where $\mathit{successor}[i]$ indicates the forum for which p_i has been captured. $\mathit{successor}[i] = \perp$ if p_i is not currently captured. It is read/write shared by all philosophers.

The algorithm, which we refer to as CTP-2, is given in Figure 3. It begins by letting p_i set its $\mathit{flag}[i]$ to $\langle \mathit{request}, \mathsf{F} \rangle$ to express its interest in F. To complete the request, p_i must also clear $\mathit{successor}[i]$ to indicate that it is not captured by any philosopher. Then, like CTP-Temp, p_i begins a repeat-loop to test if it can attend F. In line 5, in addition to the two conditions $\mathit{turn} = \mathsf{F}$ and $\mathit{all\_passive}(\overline{\mathsf{F}})$, a philosopher p_i exits the while-loop if it finds that it is a successor of some other philosopher, which is determined by the condition $\mathit{successor}[i] = \mathsf{F}$. If p_i is a successor, then it also skips the repeat-loop and enters CS to attend F. By 'CS'—the Critical Section—we mean the program segment from line 8 to line 13.

If p_i is not a successor of any other philosopher, then like CTP-Temp, it obtains a temporary permission to attend F if F has priority or no one else is interested in $\overline{\mathsf{F}}$ (line 5). To actually attend F, p_i must check if no philosopher has obtained a temporary permission to attend $\overline{\mathsf{F}}$ (by evaluating $\mathit{none\_in\_cs}(\overline{\mathsf{F}})$). In addition, p_i must also check if all successors of a previous captain interested in $\overline{\mathsf{F}}$ have finished $\overline{\mathsf{F}}$. The new predicate $\mathit{no\_successor}(\overline{\mathsf{F}})$ defined in Figure 3 serves this purpose. Note that the evaluation of $\mathit{no\_successor}(\overline{\mathsf{F}})$ involves, for each p_j, two accesses to p_j's variables: $\mathit{flag}[j]$ and $\mathit{successor}[j]$. As shall be clear in the analysis, the order of the two accesses is irrelevant to the correctness of the algorithm.

It is also important to note that the condition $\mathit{none\_in\_cs}(\overline{\mathsf{F}})$ in line 7 must be evaluated before $\mathit{no\_successor}(\overline{\mathsf{F}})$. (Throughout the paper we use $c_1 \overset{\rightarrow}{\wedge} c_2$ to

[4] The idea of capturing is distantly related to that of *helping* from the concurrent objects literature [14, 3]. In the construction of concurrent objects, some helping mechanisms are devised to let processes "help" each other to make progress so that if some process makes progress, then eventually every process does. Our capturing procedure also allows a philosopher in the critical section to "help" others to enter the critical section, but the technical details are entirely different.

/* assuming p_i is attempting F */

$\mathit{flag}[i] \leftarrow \langle \mathit{request}, \mathsf{F} \rangle$;
$\mathit{successor}[i] \leftarrow \perp$;
repeat
    $\mathit{flag}[i] \leftarrow \langle \mathit{request}, \mathsf{F} \rangle$;
    while $\mathit{successor}[i] \neq \mathsf{F} \wedge \mathit{turn} \neq \mathsf{F} \wedge \neg \mathit{all\_passive}(\overline{\mathsf{F}})$ **do** **skip** ;
    $\mathit{flag}[i] \leftarrow \langle \mathit{in\_cs}, \mathsf{F} \rangle$;
until $\mathit{successor}[i] = \mathsf{F} \vee (\mathit{none\_in\_cs}(\overline{\mathsf{F}}) \overrightarrow{\wedge} \mathit{no\_successor}(\overline{\mathsf{F}}))$;
/* beginning of critical section */
if $\mathit{successor}[i] \neq \mathsf{F}$ **then** {
    $\mathit{turn} \leftarrow \overline{\mathsf{F}}$;
    for $j \leftarrow 0$ **to** $n-1,\ j \neq i,$ **do** /* start capturing philosophers */
        if $\mathit{flag}[j] \in \{\langle \mathit{request}, \mathsf{F} \rangle, \langle \mathit{in\_cs}, \mathsf{F} \rangle\}$ **then** $\mathit{successor}[j] \leftarrow \mathsf{F}$; }
$<<$ talk in F $>>$
$\mathit{flag}[i] \leftarrow \langle \mathit{passive}, \perp \rangle$;
/* end of critical section */

where

$\mathit{all\_passive}(\overline{\mathsf{F}}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : \mathit{flag}[j] = \langle \mathit{state}, \mathit{op} \rangle \Rightarrow \mathit{op} \neq \overline{\mathsf{F}}$

$\mathit{none\_in\_cs}(\overline{\mathsf{F}}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : \mathit{flag}[j] = \langle \mathit{state}, \mathit{op} \rangle \Rightarrow (\mathit{state} \neq \mathit{in\_cs} \vee \mathit{op} = \mathsf{F})$

$\mathit{no\_successor}(\overline{\mathsf{F}}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : \mathit{flag}[j] = \langle \mathit{state}, \mathit{op} \rangle \Rightarrow \neg(\mathit{successor}[j] = \overline{\mathsf{F}} \wedge \mathit{op} = \overline{\mathsf{F}})$

Figure 3: Algorithm CTP-2.

denote the conjunction of c_1 and c_2 where c_1 is evaluated before c_2.) Otherwise, mutual exclusion cannot be guaranteed (see Section 4.4).

Finally, p_i must also set *turn* to $\overline{\mathsf{F}}$ (line 9), and then capture philosophers interested in the same forum. The latter is done by setting their *successor*s to F (lines 10-11). After exiting the forum, p_i resets its *flag* and returns to thinking.

4.3 Analysis of CTP-2

For the purpose of analysis, we formalize the terms used in the algorithm. We say that a philosopher p_i *has requested* F (or more colloquially, p_i is *interested* in F) if p_i has executed line 1 of CTP-2. To complete a request p_i must also set $successor[i]$ to $\perp$. We say that p_i has *completed* a request for F if it has executed lines 1-2. The request is *granted* when p_i exits the repeat-loop in lines 3-7. p_i is *waiting* for F if it has completed a request for F but the request has not yet been granted.

Recall that the CS region refers to the program segment from line 8 to line 13. So p_i *enters* CS when its request is granted, and *exits* CS when it finishes line 13. A *passage* through CS by p_i thus refers to a time interval from the point p_i enters CS to the point it leaves CS. Since the CS region is used to implement a forum session, "a passage through CS" will be used synonymously with "a passage through a forum" in the analysis.

In the algorithm a philosopher p_i interested in F enters CS only if

1. $successor[i] = \mathsf{F}$, or
2. $successor[i] \neq \mathsf{F}$ and $(none\_in\_cs(\overline{\mathsf{F}}) \overrightarrow{\wedge} no\_successor(\overline{\mathsf{F}})) = true$

In the first case we say that p_i enters CS as a *successor*, while in the latter case p_i enters CS as a *captain*. Observe that p_i resets $successor[i]$ to $\perp$ each time it completes a request, and will not alter it while waiting for the request to be granted. So for $successor[i]$ to be F, some philosopher must have changed $successor[i]$ to F after p_i completes its request. We say that p_j *captures* p_i if p_j writes F to $successor[i]$ while p_i is waiting for F. In this case, p_i must enter CS as a successor; p_i is then called a *successor* of p_j, and p_j a *captain* of p_i. Note that when p_i exits CS, the successor-captain relation between p_i and p_j is broken. So when we say "p_i is a successor of p_j" or "p_j is a captain of p_i", we implicitly assume a passage by p_i and a passage by p_j to which the phrase refers. Note further that since more than one philosopher may write F to $successor[i]$, p_i may have more than one captain at a time.

Since more than one philosopher may enter CS as a captain simultaneously, they may attempt to capture one another. If p_i interested in F enters CS as a captain, but before p_i executes the if-then statement in line 8 another captain p_j has written F to $successor[i]$, then p_i will not be able to capture any philosopher. In this case we say that p_i is a captain *killed* by p_j. A captain that is not killed by any other captain is called an *effective* captain. Clearly, if p_j kills p_i, then p_j must be an effective captain. Furthermore, although an effective captain is able to capture successors, it may end up with no successor if no one is interested in the same forum.

We are now ready for the analysis. We begin with mutual exclusion.

Lemma 4.1 *If p_i enters CS as a captain, then while it is in CS, no other p_j interested in a different forum can be in CS as a captain simultaneously.*

Proof. Assume p_i is interested in F. By definition of captain, p_i must enter CS with a condition $none\_in\_cs(\overline{\mathsf{F}}) = true$. Observe that p_i sets its state to in_cs before it evaluates $none\_in\_cs(\overline{\mathsf{F}})$. So when p_i inspects other philosophers' states, no p_j interested in a different forum can be in state in_cs (and so can be in CS) at this moment. Subsequently, when p_j enters state in_cs to inspect p_i's state, if p_i is still in CS, p_j must find that $none\_in\_cs(\mathsf{F}) = false$ and so cannot enter CS. □

Lemma 4.2 *If p_i enters CS as a captain, then after it leaves CS, no other p_j interested in a different forum can enter CS as a captain until all of p_i's successors have left CS.*

Proof. Assume that p_i is interested in F. For p_j to enter CS as a captain, it must evaluate both $none\_in\_cs(\mathsf{F})$ and $no\_successor(\mathsf{F})$ to true, and the first predicate must be evaluated before the second. By Lemma 4.1, to obtain $none\_in\_cs(\mathsf{F}) = true$ p_i must have left CS when p_j inspects p_i's flag in the evaluation. So when p_j evaluates $no\_successor(\mathsf{F})$, p_i must have finished capturing philosophers (lines 10-11 of the algorithm). Let p_k be a successor of p_i. For distinguishing purposes we shall use F^* to denote the instance of F for which p_k enters CS as a successor of p_i.

For p_j to evaluate $no\_successor(\mathsf{F})$ to true, the following must hold (note that no particular ordering is assumed in accessing $flag[k]$ and $successor[k]$):

$$flag[k] = \langle state, op \rangle \Rightarrow \neg(successor[k] = op = \mathsf{F})$$

So there are four cases: $successor[k] = \perp$, $successor[k] = \overline{\mathsf{F}}$, $op = \perp$, or $op = \overline{\mathsf{F}}$. Since p_i has written F^* to $successor[k]$, and since only p_k can reset $successor[k]$ to $\perp$ (which occurs only after p_k has requested another forum), the case $successor[k] = \perp$ seen by p_j implies that p_k has already finished F^* and left CS.

The case $successor[k] = \overline{\mathsf{F}}$ implies that some philosopher p_l reads $op = \overline{\mathsf{F}}$ in $flag[k]$ and writes $\overline{\mathsf{F}}$ to $successor[k]$ after p_i has written F^* to $successor[k]$. This implies that p_k has left CS in between, as a successor of p_i.

For the case $op = \perp$, recall that p_j reads $flag[k]$ after p_i writes F^* to $successor[k]$. Moreover, since p_k is a successor of p_i, p_k must have completed a request for F^* before p_i writes F^* to $successor[k]$. Then we have the following events that happen in the order listed:

- p_k sets $flag[k]$ to $\langle request, \mathsf{F}^* \rangle$ and resets $successor[k]$ to $\perp$.
- p_i writes F^* to $successor[k]$.
- p_j reads $flag[k] = \langle passive, \perp \rangle$.

Therefore, when p_j finds that $flag[k] = \langle passive, \perp \rangle$, p_k must have already finished F^* and left CS.

Finally, consider the case $op = \overline{\mathsf{F}}$. Similar to the above argument we can show that for p_j to read $flag[k] = \langle state, \overline{\mathsf{F}} \rangle$, p_k must have finished F^* and have requested $\overline{\mathsf{F}}$. The lemma is then proven. □

Lemma 4.3 *If p_i enters CS as a successor, then it cannot capture any philosopher while in CS.*

Proof. By definition of successor, p_i enters CS with the condition $successor[i] = \mathsf{F}$ (assuming p_i is interested in F). Because no philosopher can change $successor[i]$ to $\overline{\mathsf{F}}$ (Lemma 4.2), $successor[i]$ remains F in between the time p_i finishes line 7 and the time it is to execute line 8. So p_i skips lines 9-11 of CTP-2. □

Theorem 4.4 *CTP-2 guarantees mutual exclusion.*

Proof. Suppose p_i interested in F enters CS as a captain. By Lemmas 4.1 and 4.2, while p_i and its successors are in CS, no philosopher interested in $\overline{\mathsf{F}}$ can enter CS as a captain. Moreover, Lemma 4.3 implies that for a philosopher interested in $\overline{\mathsf{F}}$ to enter CS as a successor, some philosopher interested in $\overline{\mathsf{F}}$ must enter CS as a captain to capture the philosopher. So while p_i and its successors are in CS, no philosopher interested in $\overline{\mathsf{F}}$ can enter CS as a successor, either. All together, we have that while a philosopher is in CS (either as a captain or as a successor), no philosopher interested in a different forum can enter CS (either as a captain or as a successor). □

We now prove bounded delay.

Lemma 4.5 *Suppose p_j enters CS as a captain. If p_i enters CS as a successor of p_j, then while p_j stays in CS, p_i cannot re-enter CS as a successor of p_j. Similarly, if p_i enters CS as a captain killed by p_j, then when p_i re-enters CS as a captain while p_j is still in CS, p_i cannot be killed again by p_j.*

Proof. This follows from the fact that p_j in CS attempts to write $successor[i]$ only once, and p_i resets $successor[i]$ to $\perp$ before it completes a request. $\square$

The above lemma implies that a captain in a passage through F can capture/kill at most $k-1$ philosophers, where k is the number of philosophers that can potentially attend F. Moreover, since only an effective captain can capture/kill philosophers, the lemma implies that if p_i repeatedly enters CS to attend F, then either p_i or some other p_j interested in F must repeatedly enter CS as an effective captain.

Lemma 4.6 *If a philosopher is waiting for a forum, then eventually some philosopher will attend a forum.*

Proof. Suppose p_i has requested F. If no philosopher has requested $\overline{\mathsf{F}}$, then the three Boolean conditions $all\_passive(\overline{\mathsf{F}})$, $none\_in\_cs(\overline{\mathsf{F}})$, and $no\_successor(\overline{\mathsf{F}})$ evaluate to true. So p_i can exit both the while-loop and the repeat-loop of the algorithm to enter CS to attend F. So for the rest of the proof assume that some p_j has requested $\overline{\mathsf{F}}$. Moreover, since a philosopher spends only a finite amount of time in CS, and since a philosopher cannot enter CS as a successor unless some philosopher is in CS as a captain, we shall further assume that no philosopher is currently in CS, and no philosopher will enter CS as a successor.

While p_i and p_j are waiting for F and $\overline{\mathsf{F}}$ respectively, if no philosopher gets into CS, then p_i and p_j must iterate through either the while-loop or the repeat-loop of the algorithm. Since no one gets into CS, *turn* remains the same value, say F (the other case is similar). So p_i cannot be blocked in the while-loop. On the other hand, p_j and every other philosopher interested in $\overline{\mathsf{F}}$ will eventually be blocked in the while-loop because $turn = \mathsf{F}$ and $all\_passive(\mathsf{F}) = false$. Therefore, eventually p_i will evaluate $none\_in\_cs(\overline{\mathsf{F}}) \overrightarrow{\wedge} no\_successor(\overline{\mathsf{F}})$ to true, and then will exit the repeat-loop to enter CS. $\square$

Lemma 4.7 *If p_j interested in F enters CS as a captain while p_i is waiting for F, then when p_j starts to capture philosophers, either p_i will be captured by p_j, or p_i will have already entered CS.*

Proof. This follows directly from the algorithm. $\square$

By Lemma 4.7, if a set of philosophers have completed their requests for F, then if one of them gets to attend F, the others will also attend F before a different forum is established.

For the following lemma, recall Definition 2.3 that "a round of F" is a maximal set $\mathcal{S}$ of consecutive passages through F such that no passage through a different forum is interspersed among them. It is clear that to start a round of F some p_j interested in F must enter CS as a captain, and when a round of passages terminates, the next round of passages must be for a different forum.

Lemma 4.8 *Suppose p_i has completed a request for F. Then the following must hold:*

1. *If a round of F is already ongoing when p_i completes its request, then either p_i will attend F, or a round of $\overline{\mathsf{F}}$ will start.*
2. *If a round of F starts after p_i has completed the request, then p_i must make a passage through F in this round.*

Proof. The second case follows directly from Lemma 4.7 and the fact that within a round of F there must be some philosopher that enters CS as an effective captain. (In Lemma 4.7, it is easy to see that the passages through F by p_i and p_j must belong to the same round.)

For the first case, if p_i remains waiting for F, then by Lemma 4.6 either some philosopher will enter CS to start a round of $\overline{\mathsf{F}}$, or some philosopher will repeatedly enter CS to attend F. If some philosopher repeatedly attends F, then by Lemma 4.5 some philosopher interested in F must also repeatedly enter CS as an effective captain; then by Lemma 4.7 p_i will be able to attend F. So the case is proven because p_i will attend F, or a round of $\overline{\mathsf{F}}$ will start. □

Lemma 4.9 *Suppose p_i is waiting for F while a round of $\overline{\mathsf{F}}$ is ongoing. Then eventually the round of $\overline{\mathsf{F}}$ will terminate and a round of F will start.*

Proof. While p_i is waiting for F, by Lemma 4.6 either p_i or some other philosopher eventually attends F, or otherwise some philosopher must repeatedly attend $\overline{\mathsf{F}}$. By the mutual exclusion property (Theorem 4.4), the first case implies that the ongoing round of $\overline{\mathsf{F}}$ will terminate and a round of F will then start.

For the second case, by Lemma 4.5 some p_j interested in $\overline{\mathsf{F}}$ must repeatedly enter CS as an effective captain. However, this causes a contradiction because after p_j enters CS as an effective captain, it will set *turn* to F. *turn* will then remain F until some philosopher interested in F enters CS to change *turn* back to $\overline{\mathsf{F}}$. While $turn = \mathsf{F}$ and p_i remains waiting for F, p_j cannot re-enter CS as a captain. So the second case also implies that the round of $\overline{\mathsf{F}}$ will terminate and a round of F will start. $\square$

Theorem 4.10 *CTP-2 guarantees bounded delay.*

Proof. Suppose p_i has completed a request for F. Consider first that a round of F is already ongoing when p_i completes its request. Then by Case 1 of Lemma 4.8, either p_i will attend F, or a round of $\overline{\mathsf{F}}$ will start. In the former case, we are done. In the latter case, since p_i's request for F has not been granted, by Lemma 4.9 the round of $\overline{\mathsf{F}}$ will terminate and a new round of F will start. Then by Case 2 of Lemma 4.8 p_i will attend F in this new round.

Next, suppose that a round of $\overline{\mathsf{F}}$ is ongoing when p_i completes its request. Then by Lemma 4.9 the round of $\overline{\mathsf{F}}$ will eventually terminate and a round of F will start. By Case 2 of Lemma 4.8 p_i will attend F in this round.

If no round of passages is ongoing when p_i completes its request, then by Lemma 4.6 eventually some round of passages will start. If it is a round of F, then by Case 2 of Lemma 4.8 p_i will attend F in this round. If it is a round of $\overline{\mathsf{F}}$, then by Lemma 4.9 this round of $\overline{\mathsf{F}}$ will eventually terminate and a round of F will start. Case 2 of Lemma 4.8 then ensures that p_i will attend F in that round. $\square$

From the above proof, when a philosopher p_i completes a request for F, it waits for at most 2 rounds of passages before a round of F is initiated in which p_i can make a passage through F. So CTP-2's forum-switch complexity is 2. Time and concurrency will be analyzed in Section 5.3 when we extend the algorithm to m fora.

Finally, it is easy to see that if no philosopher is interested in $\overline{\mathsf{F}}$, then every philosopher attempting to attend F can do so, and they can attend F concurrently. The other case that philosophers can attend $\overline{\mathsf{F}}$ concurrently is similar. So CTP-2 allows concurrent entering.

4.4 Remarks

We comment here on some code of CTP-2 relating to its correctness and performance. First, as noted in Section 4.2, the condition $\mathit{none\_in\_cs}(\overline{\mathsf{F}})$ in line 7 of CTP-2 must be evaluated before $\mathit{no\_successor}(\overline{\mathsf{F}})$, for otherwise mutual exclusion cannot be guaranteed. To see this, assume that p_i and p_j wish to attend F, while p_k wishes to attend $\overline{\mathsf{F}}$. Assume further that $\mathit{turn} = \mathsf{F}$. Consider the following scenario:

1. p_k sees that no philosopher is interested in F, and so it exits the while-loop in line 5.

2. p_i also exits the while-loop because $turn = \mathsf{F}$. It then sets $flag[i]$ to $\langle in\_cs, \mathsf{F} \rangle$, finishes line 7, and enters CS.

3. p_j sets its flag to $\langle request, \mathsf{F} \rangle$ and proceeds to line 5.

4. p_k changes its state to in_cs and starts to evaluate the conditions in line 7. Suppose $no\_successor(\mathsf{F})$ is evaluated first. Since no philosopher interested in F is captured as a successor, p_k sees that $no\_successor(\mathsf{F}) = true$. p_k then evaluates $none\_in\_cs(\mathsf{F})$. It begins with p_j and finds that p_j is in state $request$. So p_k continues to check p_i's flag.

5. Before p_k inspects p_i's flag, p_i finds that p_j is also interested in F and so it captures p_j. p_i then sets $turn$ to $\overline{\mathsf{F}}$, finishes F, and resets its state to $passive$.

6. p_k now sees that p_i's state is passive and so obtains $none\_in\_cs(\mathsf{F}) = true$. It then exits line 7 and enters CS to attend $\overline{\mathsf{F}}$.

7. p_j in line 5 learns that it is captured as a successor and so moves on to attend F, yielding both F and $\overline{\mathsf{F}}$ to be in session simultaneously.

Note that if $none\_in\_cs(\mathsf{F})$ is evaluated first, then p_i must have already captured p_j when p_k sees $none\_in\_cs(\mathsf{F}) = true$. So when p_k evaluates $no\_successor(\mathsf{F})$, it cannot return true unless p_j has finished F.

Second, the statement "$turn \leftarrow \overline{\mathsf{F}}$" in line 9 can be moved to the end of line 11 where p_i has finished capturing philosophers, or it can even be placed outside the if-then statement so that every philosopher entering CS will set $turn$ to $\overline{\mathsf{F}}$ (which, of course, may result in many redundant assignments to $turn$). It is not difficult to see that these modifications cannot affect the correctness of the algorithm. However, we have deliberately placed the statement before the capturing procedure to achieve optimal performance.

Intuitively, if $turn$ is changed earlier, then fewer philosophers get a chance to concurrently attend an ongoing forum, and so philosophers interested in a different forum wait for less time before attending the forum. So the choice of placing "$turn \leftarrow \overline{\mathsf{F}}$" before or after the capturing procedure should be a matter of trade-off between the algorithm's time complexity and its concurrency. However, as we shall see in Section 5.4, placing "$turn \leftarrow \overline{\mathsf{F}}$" ahead improves the algorithm's time complexity, but does not affect its degree of concurrency. (This is because the philosophers interested in F will still be able to attend F concurrently as they will be captured by the captain executing "$turn \leftarrow \overline{\mathsf{F}}$".)

Finally, although after p_i has set *flag*$[i]$ to $\langle \textit{request}, \mathsf{F} \rangle$ in line 1 it will again execute the same assignment immediately after it enters the repeat-loop, line 1 cannot be removed. This is because the capturing procedure in line 11 involves, for each p_j, a read from *flag*$[j]$ to see if p_j is interested in F and, if so, a write to *successor*$[j]$. Between the read and the write, p_j could have also entered CS as a captain, finished F, changed *flag*$[j]$ to $\langle \textit{passive}, \perp \rangle$, and then requested another entry to F. If we were to remove line 1 from the algorithm, then *flag*$[j]$ remains $\langle \textit{passive}, \perp \rangle$ after p_j has executed line 2. When p_i finally writes F to *successor*$[j]$ and leaves CS, another p_k interested in $\overline{\mathsf{F}}$ may read $\textit{turn} = \overline{\mathsf{F}}$, pass line 5, see $\textit{none\_in\_cs}(\mathsf{F}) \overrightarrow{\wedge} \textit{no\_successor}(\mathsf{F}) = \textit{true}$, and then enter CS to attend $\overline{\mathsf{F}}$. Since $\textit{successor}[j] = \mathsf{F}$, p_j can also enter CS to attend F, thus violating mutual exclusion.

5 A Generalized Solution

We now generalize CTP-2 to m fora $\mathsf{F}_0, \mathsf{F}_1, \ldots, \mathsf{F}_{m-1}$.

5.1 The Algorithm

The generalized algorithm, which we refer to as CTP-m, is given in Figure 4. Like CTP-2, a philosopher p_i completes its request for F_k by changing *flag*$[i]$ to $\langle \textit{request}, \mathsf{F}_k \rangle$ and resetting *successor*$[i]$ to $\perp$. In CTP-2, p_i must wait until (1) some captain captures p_i, (2) $\textit{turn} = \mathsf{F}_k$, or (3) no philosopher is interested in a different forum. In CTP-m, however, care must be taken to avoid deadlocks. For example, suppose two philosophers p_1 and p_2 wish to attend F_1 and F_2 respectively, and *turn* is set to a third forum, say F_0. If each p_j $(j = 1, 2)$ loops on the condition "$\textit{turn} \neq \mathsf{F}_j$" $\wedge$ "some philosopher is interested in a different forum", then both p_1 and p_2 would be waiting forever. Note that the latter condition cannot be weakened to "some philosopher is interested in the forum specified by *turn*" either. This is because then both p_1 and p_2 may find that no one is interested in F_0 and then attempt to establish F_1 and F_2 simultaneously.

To resolve this dilemma, we let the philosopher whose requesting forum is the "closest" to the one dictated by *turn* proceed. This is determined by the function $\textit{next\_op}(\mathsf{F}_g)$, which checks all philosophers' *flag*s to see if any philosopher has requested a forum. If so, $\textit{next\_op}(\mathsf{F}_g)$ returns the first requested forum in the sequence $\mathsf{F}_g, \mathsf{F}_{g+1}, \ldots, \mathsf{F}_{g+m-1}$. Otherwise, $\textit{next\_op}(\mathsf{F}_g)$ returns F_g. (Note that throughout this paper unless stated otherwise addition and subtraction on indices of F are modulo m. Moreover, if $h < g$ then $\mathsf{F}_g, \mathsf{F}_{g+1}, \ldots, \mathsf{F}_h$

```
/* assuming p_i is attempting F_k */
flag[i] ← ⟨request, F_k⟩;
successor[i] ← ⊥;
repeat
    flag[i] ← ⟨request, F_k⟩;
    while successor[i] ≠ F_k ∧ next_op(turn) ≠ F_k do skip ;
    flag[i] ← ⟨in_cs, F_k⟩;
until successor[i] = F_k
    ∨ (none_in_cs(‾F_k) →∧ no_successor(‾F_k) →∧ (turn = F_k ∨ all_passive(turn)));
/* beginning of critical section */
if successor[i] ≠ F_k then {
    turn ← next_op(F_{k+1});
    for j ← 0 to n − 1, j ≠ i, do /* start capturing philosophers */
        if flag[j] ∈ {⟨request, F_k⟩, ⟨in_cs, F_k⟩} then successor[j] ← F_k; }
<< talk in F_k >>
flag[i] ← ⟨passive, ⊥⟩;
/* end of critical section */
```

where

$all\_passive(\mathsf{F}_g) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : flag[j] = \langle state, op\rangle \Rightarrow op \neq \mathsf{F}_g$

$none\_in\_cs(\overline{\mathsf{F}_k}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : flag[j] = \langle state, op\rangle \Rightarrow (state \neq in\_cs \vee op = \mathsf{F}_k)$

$no\_successor(\overline{\mathsf{F}_k}) \equiv \forall j, j \neq i, 0 \leq j \leq n-1 : flag[j] = \langle state, op\rangle \Rightarrow$
$\neg(\exists\, l, l \neq k : successor[j] = \mathsf{F}_l \wedge op = \mathsf{F}_l)$

/* function $next\_op(\mathsf{F}_g)$ returns the first forum F_h in the sequence $\mathsf{F}_g, \mathsf{F}_{g+1}, \ldots,$ */
/* F_{g+m-1} such that some philosopher has requested F_h but no philosopher has */
/* requested $\mathsf{F}_g, \mathsf{F}_{g+1}, \ldots, \mathsf{F}_{h-1}$. Note that since $flag[i]$ is also inspected, and since */
/* p_i invokes $next\_op$ only when it is interested in F_k, $next\_op(\mathsf{F}_g)$ returns F_k if no */
/* philosopher is interested in a different forum. */

```
next_op (F_g) :: {
    next ← g + m;
    for j ← 0 to n − 1 do {
        let flag[j] = ⟨state, op⟩;
        if op ≠ ⊥ then {
            let op = F_l;
            if l < g then l ← l + m;
            if l < next then next ← l; }}
    return F_{next (mod m)};
}
```

Figure 4: Algorithm CTP-m.

stands for the sequence $\mathsf{F}_g, \mathsf{F}_{g+1}, \ldots, \mathsf{F}_{m-1}, \mathsf{F}_0, \mathsf{F}_1, \ldots, \mathsf{F}_h$.) So the exit condition $\mathit{next\_op}(\mathit{turn}) = \mathsf{F}_k$ in line 5 means that from p_i's observation F_k is the requested forum that is the closest to the one specified by *turn*.

When p_i exits the while-loop, it changes $\mathit{flag}[i]$ to $\langle \mathit{in\_cs}, \mathsf{F}_k \rangle$. In CTP-2, p_i exits the repeat-loop in line 7 only if it is a successor, or no philosopher interested in a different forum is in state *in_cs* and all successors interested in a different forum have left CS. In CTP-m, if p_i is not a successor, we additionally require p_i to check if $\mathit{turn} = \mathsf{F}_k$ or no philosopher is interested in the forum dictated by *turn*. (Note that the CS region of CTP-m refers to the program segment from line 8 to line 13.) As we shall see in Sections 5.2 and 5.4, this extra condition together with our way of assigning turns guarantees that a philosopher waits for at most m rounds of passages before its request is granted. Removing this condition (i.e., $\mathit{turn} = \mathsf{F}_k \vee \mathit{all\_passive}(\mathit{turn})$) from the algorithm explodes the forum-switch complexity from $O(m)$ to $O((\frac{1+\sqrt{5}}{2})^m)$!

Like CTP-2, when p_i enters CS, if it is not a successor then it must act as a captain to capture philosophers interested in the same forum to enter CS. Then, it must give the turn to other philosophers that are waiting for a different forum. In CTP-m, the new turn is calculated by the function $\mathit{next\_op}(\mathsf{F}_{k+1})$, which assigns *turn* to the first forum in the sequence $\mathsf{F}_{k+1}, \mathsf{F}_{k+2}, \ldots, \mathsf{F}_{k+m-1}$ for which some philosopher is waiting, or F_k otherwise.

Note that unlike CTP-2 where a captain always yields *turn* to the other forum, in CTP-m a captain may set *turn* to the same forum it has requested if it sees that no one is interested in a different forum. In the presence of concurrency, this may cause *turn*, which has been set to F_h by some captain p_i (because p_i found some philosopher interested in F_h), to be reset to F_k by a slow captain p_j (because p_j evaluated $\mathit{next\_op}(\mathsf{F}_{k+1})$ earlier than p_i and found no philosopher interested in a forum other than F_k). As a result, more philosophers can enter CS to attend F_k before *turn* is finally set to F_h to allow other philosophers to attend F_h. However, as we shall see in Section 5.4, the time a philosopher may wait for its request is only slightly affected (at most by a constant factor), while the degree of concurrency is increased by $O(n)$. Therefore, in CTP-m we have opted for a higher degree of concurrency by allowing a captain to set *turn* to the same forum it has requested.

Moreover, one may have observed that the evaluation of $\mathit{next\_op}(\mathit{turn}) \neq \mathsf{F}_k$ in line 5 involves an access to *turn*, and then the inspection of philosophers' *flag*s. Thus it is possible that while the inspection is ongoing *turn* has been changed several times already, and so a philosopher may exit the while-loop even if its requesting forum is not the closest to the one currently specified by *turn*. However, the premature exit of the while-loop is not harmful because when the philosopher proceeds to line 7, it will learn that either some philosopher is already in CS or it does not have the priority to enter CS when it evaluates "$turn = \mathsf{F}_k \vee all\_passive(turn)$", and so it will go back to line 5 to re-evaluate $next\_op(turn)$, which will then bring up the more up-to-date value of *turn*.

The evaluation of "$turn = \mathsf{F}_k \vee all\_passive(turn)$" in line 7 also involves an access to *turn* and the inspection of philosophers' *flag*s. Unlike the situation in line 5, *turn* cannot be changed by any philosopher interested in a different forum during the evaluation. This is because the evaluation takes place only after $none\_in\_cs(\overline{\mathsf{F}_k}) \overrightarrow{\wedge}\ no\_successor(\overline{\mathsf{F}_k}) = true$, which implies that no philosopher interested in a different forum can be in CS to change *turn*. Note that *turn* may be changed by some p_j interested in F_k during the evaluation; but this cannot cause a problem as no philosopher interested in a different forum will be able to enter CS until p_i has exited CS.

5.2 Mutual Exclusion and Bounded Delay of CTP-m

We now prove the correctness of CTP-m. Since all the terms defined in Section 4.3 can be easily generalized to m fora, we shall use them directly in the analysis.

Theorem 5.1 *CTP-m guarantees mutual exclusion.*

Proof. This can be proved similarly to Theorem 4.4 and observe that only the exit condition

$$(successor[i] = \mathsf{F}_k) \vee (none\_in\_cs(\overline{\mathsf{F}_k}) \overrightarrow{\wedge}\ no\_successor(\overline{\mathsf{F}_k}))$$

of the repeat-loop of CTP-m suffices to guarantee mutual exclusion. □

We move on to prove that CTP-m guarantees bounded delay. For this we need the following lemmas.

Lemma 5.2 *Suppose p_j enters CS as a captain. If p_i enters CS as a successor of p_j, then while p_j stays in CS, p_i cannot re-enter CS as a successor of p_j. Similarly, if p_i enters CS as a captain killed by p_j, then when p_i re-enters CS as a captain while p_j is still in CS, p_i cannot be killed again by p_j.*

Proof. Like Lemma 4.5, this is because p_j in CS attempts to write $successor[i]$ only once, and p_i resets $successor[i]$ to $\perp$ before it completes a new request. □

Lemma 5.3 *If a philosopher is waiting for a forum, then eventually some philosopher will attend a forum.*

Proof. We shall only outline the main idea of the proof; the rest is similar to Lemma 4.6. Assume that no philosopher is currently in CS and no philosopher will enter CS as a successor. So *turn* remains unchanged. Observe that when one or more philosophers attempt to attend a forum, not all of them can be blocked in the while-loop of CTP-m. This is because a philosopher p_i whose request F_k is the closest to *turn* (w.r.t. the ordering $turn, \ldots, \mathsf{F}_j, \mathsf{F}_{j+1}, \ldots$) will obtain $next\_op(turn) = \mathsf{F}_k$ in line 5, and so will proceed to line 7. Since F_k is the closest to *turn*, the condition $turn = \mathsf{F}_k \vee all\_passive(turn)$ must hold. So if the evaluation of the Boolean condition in line 7 returns *false*, then some p_j interested in a different forum must have set its state to *in_cs* and is also evaluating the Boolean condition in line 7. (p_j must have seen a relatively old version of *turn* when it evaluated $next\_op(turn)$ in line 5.) Then both p_i and p_j will loop back to line 4. When they proceed to line 5, still, p_i will not be blocked in line 5, but p_j this time will see a correct version of *turn* and so will learn that its requesting forum is not the closest to *turn*, and so will be waiting in line 5. Clearly, at most $n-1$ philosophers can cause such a conflict situation with p_i, and all of them will eventually be blocked in line 5, after which p_i will be able to exit the loop condition in line 7 to enter CS. □

Lemma 5.4 *Suppose p_i has completed a request for F_k. Then the following must hold:*

1. *If a round of F_k is already ongoing when p_i completes its request, then either p_i will attend F_k, or a different round of passages will start.*
2. *If a round of F_k starts after p_i has completed the request, then p_i must make a passage through F_k in this round.*

Proof. The proof is similar to Lemma 4.8; we omit the details. □

Lemma 5.5 *Suppose a round of F_h is ongoing while p_i is waiting for F_k, $k \neq h$. Then the round of F_h will eventually terminate and a new round will start.*

Proof. The proof is similar to Lemma 4.9, and observe that while p_i is waiting for F_k, eventually some effective captain interested in F_h must assign *turn* to one of the fora $\mathsf{F}_{h+1}, \mathsf{F}_{h+2}, \ldots, \mathsf{F}_k$. Then every philosopher attempting to attend F_h will find $next\_op(turn) \neq \mathsf{F}_h$ when it evaluates the while-loop condition of CTP-m, and so can no longer enter CS as a captain in this round of F_h. □

Lemma 5.6 *Suppose a round of* F_l *starts while* p_i *is waiting for* F_k, $l \neq k$. *Then when the round terminates, the next round must be a round of* F_g *for some* F_g *in* $\mathsf{F}_{l+1}, \mathsf{F}_{l+2}, \ldots, \mathsf{F}_k$.

Proof. Since p_i requests F_k before the round of F_l starts, when any captain in the round calls the function $next\_op(\mathsf{F}_{l+1})$ to assign the next turn, it must obtain one of the fora $\mathsf{F}_{l+1}, \mathsf{F}_{l+2}, \ldots, \mathsf{F}_k$. So right after the round terminates, $turn = \mathsf{F}_g$ for some F_g in $\mathsf{F}_{l+1}, \mathsf{F}_{l+2}, \ldots, \mathsf{F}_k$. Moreover, some philosopher must have requested F_g during the execution of $next\_op(\mathsf{F}_{l+1})$. By the mutual exclusion property, the philosopher must still be waiting for F_g when the round of F_l terminates. So right after the round of F_l terminates, we have that (1) $turn = \mathsf{F}_g$, and (2) some philosopher is waiting for F_g.

To complete the proof of the lemma, we argue that the next round of passages must be a round of F_g. Observe that to start a new round of passages, say a round of F_h, some philosopher p_j must enter CS as a captain, and so it must exit the repeat-loop of CTP-m with true on the following condition:

$$none\_in\_cs(\overline{\mathsf{F}_h}) \overset{\rightarrow}{\wedge} no\_successor(\overline{\mathsf{F}_h}) \overset{\rightarrow}{\wedge} (turn = \mathsf{F}_h \vee all\_passive(turn))$$

Note that p_j can evaluate "$turn = \mathsf{F}_h \vee all\_passive(turn)$" only after the predicate "$none\_in\_cs(\overline{\mathsf{F}_h}) \overset{\rightarrow}{\wedge} no\_successor(\overline{\mathsf{F}_h})$" evaluates to true. That is, only after all captains and their successors in the current round of F_l have left CS. Since while p_j is evaluating "$turn = \mathsf{F}_h \vee all\_passive(turn)$" no philosopher can be in CS to change $turn$, $turn$ remains F_g during the evaluation. So if $\mathsf{F}_h \neq \mathsf{F}_g$, then $all\_passive(turn)$ must evaluate to false because some philosopher has already requested F_g before the evaluation. So no philosopher can start a round of F_h unless $\mathsf{F}_h = \mathsf{F}_g$. □

Lemma 5.7 *After* p_i *completes a request for* F_k, *at most* m *rounds of passages can occur before a round of* F_k *starts.*

Proof. The lemma holds trivially if no round of passages starts after p_i completes its request. So let us assume that some round of F_h starts after p_i completes its request. Clearly, the lemma holds if $h = k$.

If $h \neq k$, then by Lemma 5.6 at most $m - 2$ more rounds of passages can occur before a round of F_k starts. Including the round of F_h and the round of passages that might already be ongoing when p_i completes its request, therefore, at most m rounds of passages can occur before a round of F_k start. □

Theorem 5.8 *CTP-m guarantees bounded delay.*

Proof. Assume p_i has completed a request for F_k. While p_i is waiting for F_k, if no round of passages is ongoing, then by Lemma 5.3 some round of passages will start. If this is a round of F_k, then by Case 2 of Lemma 5.4, p_i must make a passage through F_k in this round. If this is not a round of F_k, then by Lemmas 5.5 and 5.7 some round of F_k eventually starts; and by Case 2 of Lemma 5.4 p_i must make a passage through F_k in this round. So p_i eventually attends F_k if no round of passages is ongoing when it completes its request.

Next, suppose some round F_h is already ongoing when p_i completes its request. If $h = k$, then by Case 1 of Lemma 5.4 eventually either p_i makes a passage through F_k or a new round of passages will start. If $h \neq k$, then by Lemma 5.5 a new round of passages will also start. Together with the previous argument, we conclude that if some round of passages is already ongoing when p_i completes its request, then p_i eventually attends F_k. □

By Lemmas 5.5 and 5.7 we can see that after p_i completes its request for F_k, it waits for at most m rounds of passages before a round of F_k starts in which p_i can make a passage through F_k. So CTP-m's forum-switch complexity is m. Of the m rounds of passages p_i has been waiting, one of them must start before p_i completes its request, and the remaining $m-1$ rounds must be of different fora.

5.3 Time Complexity and Concurrency of CTP-m

We now analyze CTP-m's time complexity and its concurrency. Recall that "passages" are represented as non-zero length intervals. Since the analysis involves handling of passages, we begin with some definitions on intervals.

Let U be a (closed) interval, and let $s(U)$ and $e(U)$ denote its start point and end point, respectively. By $\|U\|$ we mean the length of U, i.e., $\|U\| = e(U) - s(U)$, and by $t \in U$ we mean $s(U) \leq t \leq e(U)$. The *intersection* of U and V, denoted by $U \sqcap V$, is defined to be the maximum interval W such that $\forall t \in W : t \in U \wedge t \in V$, or $\perp$ otherwise. Two intervals U and V *overlap* if $\|U \sqcap V\| > 0$. The *projection* of U in $[t_s, t_f]$, denoted by $U|_{t_s}^{t_f}$, is the interval $U \sqcap [t_s, t_f]$ if U and $[t_s, t_f]$ overlap, or $\perp$ otherwise. If $\mathcal{S}$ is a set of intervals, then $\mathcal{S}|_{t_s}^{t_f} = \{U|_{t_s}^{t_f} : U \in \mathcal{S},\ U|_{t_s}^{t_f} \neq \perp\}$.[5]

For the proofs in this section, it is useful to recall Definition 2.2. The following proposition follows directly from the above definitions.

[5]Since we allow passages to occur concurrently, in the paper, unless stated otherwise, all sets consisting of intervals are treated as *multisets*.

Proposition 5.9 *Let $\mathcal{S}$ be a set of intervals. Then for any $t, t_s \leq t \leq t_f$, $\mathbf{dim}(\mathcal{S}|_{t_s}^{t_f}) \leq \mathbf{dim}(\mathcal{S}|_{t_s}^{t}) + \mathbf{dim}(\mathcal{S}|_{t}^{t_f})$. Moreover, if $t \notin U$ for all $U \in \mathcal{S}$, then $\mathbf{dim}(\mathcal{S}|_{t_s}^{t_f}) = \mathbf{dim}(\mathcal{S}|_{t_s}^{t}) + \mathbf{dim}(\mathcal{S}|_{t}^{t_f})$.*

As analyzed in the previous section, after p_i completes a request for F_k, it waits for at most m rounds of passages before a round of F_k is initiated in which p_i can make a passage through F_k. To analyze the time complexity, we first determine the number of passages that may occur in each round. This is done in the following lemma. Of course, if the round starts before p_i completes its request, then we are only concerned with the passages that occur after p_i completes its request. The time t_s set up in the following lemma is for this purpose. Moreover, n_h denotes the number of philosophers that can potentially attend F_h.

Lemma 5.10 *Suppose a round of F_h is ongoing while p_i is waiting for F_k, $\mathsf{F}_k \neq \mathsf{F}_h$. Let $t_s = \max(t_{s_1}, t_{s_2})$, where t_{s_1} is the time the round of F_h starts, and t_{s_2} the time p_i completes its request for F_k, and let t_f be the time the round of F_h terminates. Furthermore, let $\mathcal{S}_h$ be the set of passages that may overlap with the interval $[t_s, t_f]$. Then*

$$|\mathcal{S}_h| \leq \frac{n_h(n_h+1)(2n_h+1)}{12} + \frac{3n_h(n_h+1)}{4}$$

Proof. We begin by distinguishing between three types of passages through CS:

- α-passage: the philosopher making this passage sets *turn* to the same forum as its request.
- β-passage: the philosopher making this passage sets *turn* to a forum different from its request.
- γ-passage: the philosopher making this passage is unable to set *turn*.

By the algorithm, a philosopher that makes an α- or β-passage must enter CS as an effective captain, while a γ-passage must be made by a successor or a killed captain.

Note that since we are only concerned with passages that overlap with the interval $[t_s, t_f]$, i.e., passages in $\mathcal{S}_h$, unless stated otherwise, all passages considered in the proof belong to $\mathcal{S}_h$. By the mutual exclusion property of CTP-m, the philosophers making these passages are all interested in F_h.

Observe that an α-passage cannot be initiated after p_i has requested F_k. This is because if p_j enters CS after p_i has requested F_k, then when p_j computes *next_op*(F_{h+1}) in line 9 of CTP-m, it must obtain a forum with an index other than h.

Moreover, when some p_1 completes a β-passage, no other p_2 can initiate a β-passage unless another philosopher has made an α-passage to reset *turn* to F_h.[6] To see this, suppose that p_1 in its β-passage sets *turn* to F_g. If p_2 completes its F_h request before p_1 sets *turn* to F_g, then when p_1 starts capturing philosophers, either p_2 will be captured by p_1 (and thus p_2 can only make a γ-passage for its request), or p_2 must have already initiated a β-passage (and thus p_2's β-passage cannot be initiated after p_1 completes its β-passage). If p_2 completes its request after p_1 has set *turn* to F_g, then when p_2 starts to evaluate the condition "$turn = \mathsf{F}_h \vee all\_passive(turn)$" in line 7, it must obtain *false* and so cannot make a β-passage.

So in the absence of α-passages all β-passages must overlap. Let $q_1, q_2, \ldots, q_l$ be the philosophers that are making these overlapping β-passages and, without loss of generality, assume that q_1 completes its passage earlier (or at least no later) than q_2, which completes its passage earlier than q_3, and so on. Suppose that q_1 in its β-passage can capture at most r philosophers (and recall that none of them can be captured more than once in the passage). Then, q_1's β-passage can result in at most r γ-passages. Moreover, none of these r philosophers involves $q_2, \ldots, q_l$ (and q_1) because when q_1 captures these r philosophers the philosophers $q_2, \ldots, q_l$ are still in their β-passages. (Recall from the previous discussion that $q_2, \ldots, q_l$ must have already initiated their β-passages when q_1 starts to capture philosophers, and they cannot complete their β-passages until q_1 has finished capturing philosophers because q_1 completes its β-passage earlier than they do.)

Because after q_1 completes its β-passage it may re-enter CS as a successor of q_2, and because after completing their γ-passages the r philosophers captured by q_1 may also re-enter CS as a successor of q_2, q_2's β-passage can result in at most $(r+1)$ γ-passages. In general, q_i's β-passage can result in at most $(r+i-1)$ γ-passages. So the l β-passages overall can result in at most

$$\sum_{1 \le i \le l} r + i - 1 = l \cdot r + \frac{l(l-1)}{2}$$

γ-passages. Together with the l β-passages, the total number of passages they can generate is at most

$$l \cdot r + \frac{l(l-1)}{2} + l = l \cdot r + \frac{l(l+1)}{2}$$

Given that $l + r = n_b$ for some $n_b \le n_h$, the total number of passages these n_b philosophers can generate is at most $n_b(n_b+1)/2$.

[6] Note that this property would not hold if line 9 is placed *after* line 11; that is, if a captain sets *turn* after it has finished capturing philosophers. See Section 5.4.

If some α-passage has been completed between these β- and γ-passages, then a new set of overlapping β-passages may be initiated.[7] Since the philosopher completing the α-passage may later re-enter CS to initiate a β-passage, by the above argument, a total number of n_b+1 philosophers can be involved in making a new series of β- and γ-passages, resulting in at most $(n_b+1)(n_b+2)/2$ more passages. Note, however, that before these (n_b+1) philosophers initiate the new series of β- and γ-passages, the philosopher making the α-passage may first capture the n_b philosophers to initiate an additional number of n_b γ-passages. So overall they can result in $(n_b+1)(n_b+2)/2+n_b$ passages.

Similarly, if after the second series of β- and γ-passages another α-passage has been completed, then a third series of at most $(n_b+2)(n_b+3)/2+(n_b+1)$ of β- and γ-passages is possible. Note that since no philosopher can initiate an α-passage after p_i has requested F_k, the previous α-passage and the current one must be made by different philosophers.

Suppose that there are n_a ongoing α-passages when p_i completes its request for F_k. Then, including the first series of $\frac{n_b(n_b+1)}{2}$ β- and γ-passages, these α-passages overall can generate at most

$$\frac{n_b(n_b+1)}{2}+\sum_{1\le i\le n_a}\left(\frac{(n_b+i)(n_b+i+1)}{2}+n_b+i-1\right)$$

of β- and γ-passages. Recall that n_b is the maximum number of philosophers that can be involved in the first series of β- and γ-passages. Clearly, none of these n_b philosophers can make any of the n_a α-passages. So $n_a+n_b\le n_h$. Therefore, the total number of α-, β-, and γ-passages that can be initiated after p_i completes a request for F_k but before the current round of F_h terminates, plus the number of passages that may be already ongoing when p_i completes its request (which, in the above discussion, are the n_a α-passages), is at most

$$\begin{aligned}&\frac{n_b(n_b+1)}{2}+\sum_{1\le i\le n_a}\left(\frac{(n_b+i)(n_b+i+1)}{2}+n_b+i-1\right)+n_a\\ \le\ &\frac{n_h(n_h+1)(2n_h+1)}{12}+\frac{3n_h(n_h+1)}{4}\end{aligned}$$

□

[7]The philosopher making this α-passage must have obtained $\mathit{next\_op}(\mathsf{F}_{h+1})=\mathsf{F}_h$ in line 9 before p_i completes its request for F_k (because it does not find any philosopher interested in a different forum), but have not yet assigned F_h to *turn* until $q_1,\ldots,q_l$ set *turn* to a different forum in their β-passages.

Note that the above bound for $|\mathcal{S}_h|$ is tight because we can construct a scenario to reach this bound. To illustrate, consider the following scenario for $n_h = 3$:

1. Each of p_1, p_2, p_3, one after another, initiates an α-passage as follows: it requests F_h, enters CS, finds that no one is interested in a different forum, and so proceeds to obtain $\mathit{next\_op}(\mathsf{F}_{h+1}) = \mathsf{F}_h$.
2. p_0 completes a request for F_k.
3. p_1 sets *turn* to F_h and then completes its α-passage.
4. p_1 makes a β-passage as follows: it requests F_h, enters CS, sets *turn* to F_k, and exits CS.
5. p_2 sets *turn* to F_h, captures p_1 (after p_1 has requested another entry to F_h), and then completes its α-passage.
6. p_1, as a successor of p_2, makes a γ-passage.
7. p_1 and p_2 respectively initiate a β-passage as follows: each requests F_h, enters CS, and proceeds to capture philosophers.
8. p_1 captures none, sets *turn* to F_k, and completes its β-passage.
9. p_2 captures p_1 (after p_1 has requested another entry to F_h), sets *turn* to F_k, and completes its β-passage.
10. p_1, as a successor of p_2, makes a γ-passage.
11. p_3 sets *turn* to F_h, captures p_1 and p_2 (after they have requested a new F_h), and then completes its α-passage.
12. p_1 and p_2, as a successor of p_3, make a γ-passage respectively.
13. p_1, p_2, and p_3 respectively initiate a β-passage as follows: each requests F_h, enters CS, and proceeds to capture philosophers.
14. p_1 captures none, sets *turn* to F_k, and completes its β-passage.
15. p_2 captures p_1 (after p_1 has requested another entry to F_h), sets *turn* to F_k, and completes its β-passage.
16. p_1, as a successor of p_2, makes a γ-passage.
17. p_3 captures p_1 and p_2 (after they have requested another entry to F_h), sets *turn* to F_k, and completes its β-passage.
18. Each of p_1 and p_2, as a successor of p_3, makes a γ-passage.

Therefore, 16 passages (3 α-passages, 6 β-passages, and 7 γ-passages) have passed after p_0 completes a request for F_k.

Next, we compute the maximum dimension of the set of passages that may occur in a round.

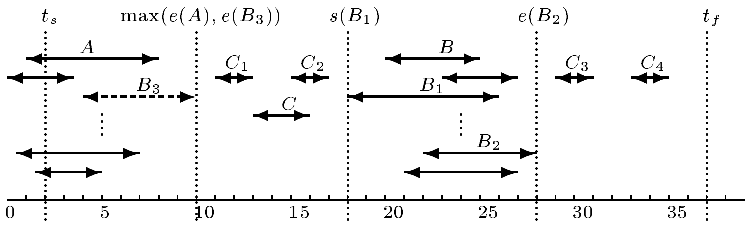

Figure 5: Layout of the passages in $\mathcal{S}_h$.

Lemma 5.11 *Suppose a round of* F_h *is ongoing while* p_i *is waiting for* F_k*,* $\mathsf{F}_k \neq \mathsf{F}_h$*. Let* $t_s = \max(t_{s_1}, t_{s_2})$*, where* t_{s_1} *is the time the round of* F_h *starts, and* t_{s_2} *the time* p_i *completes its request for* F_k*, and let* t_f *be the time the round of* F_h *terminates. Furthermore, let* $\mathcal{S}_h$ *be the set of passages that overlap with the interval* $[t_s, t_f]$*. Then*

$$\mathbf{dim}(\mathcal{S}_h) \leq 2n_h + 3$$

Proof. Let α-, β-, and γ-passages be defined as in Lemma 5.10. Let $\mathcal{A}$ be the set of α-passages in $\mathcal{S}_h$, and let A be the passage in $\mathcal{A}$ with the largest end time. Furthermore, let $\mathcal{B}$ be the set of β-passages in $\mathcal{S}_h$ that are initiated no earlier than $e(A)$; i.e., $\forall U \in \mathcal{B} : s(U) \geq e(A)$. Recall from Lemma 5.10 that all α-passages in $\mathcal{A}$ must be initiated before p_i requests F_k, and that all β-passages in $\mathcal{B}$ must overlap (at a common point).

Let B_1 be the passage, among the passages in $\mathcal{B}$, with the smallest start time, and B_2 be the passage with the largest end time. Since B_1 and B_2 overlap, they constitute a cover of $\mathcal{S}_h|_{s(B_1)}^{e(B_2)}$ (see Figure 5).

Assume first that there is some β-passage that starts before $e(A)$. Let B_3 be such a passage with the largest end time. Assume further that $e(B_3) < s(B_1)$. So there is no β-passage overlapping with $[e(B_3), s(B_1)]$ (because for every β-passage $B' \in \mathcal{S}_h$, if $s(B') < e(A)$ then $e(B') \leq e(B_3)$, and if $s(B') \geq e(A)$ then $s(B') \geq s(B_1)$). Since A is the passage in $\mathcal{A}$ with the largest end time, there is no α- and β-passage overlapping with $[\max(e(A), e(B_3)), s(B_1)]$.

Consider the maximum number of γ-passages that can be interspersed between $[\max(e(A), e(B_3)), s(B_1)]$. Let C_1 and C_2 be any two γ-passages that overlap with the interval $[\max(e(A), e(B_3)), s(B_1)]$. (See Figure 5 again.) We argue that C_1 and C_2 cannot be made by the same philosopher. This is because if C_1 and C_2 were made by the same philosopher, say p_j, then C_1 and C_2 must not overlap. Without loss of generality assume that $e(C_1) < s(C_2)$. Then some captain, while making an α- or β-passage, must have captured or killed p_j in $[e(C_1), s(C_2)]$. This then contradicts the fact that no α- and β-passage may overlap with $[\max(e(A), e(B_3)), s(B_1)]$.

Furthermore, let U be one of the two passages A and B_3 with the largest end time. Since no philosopher makes an α- or β-passage in $[\max(e(A), e(B_3)), s(B_1)]$, after a philosopher completes U, it cannot make a γ-passage in the interval. So no γ-passage overlapping with $[\max(e(A), e(B_3)), s(B_1)]$ can be made by the philosopher that completes U.

The above argument implies that at most $(n_h - 1)$ γ-passages can overlap with $[\max(e(A), e(B_3)), s(B_1)]$. Since no α- or β-passage can overlap with $[\max(e(A), e(B_3)), s(B_1)]$, a minimal cover of $\mathcal{S}_h|_{\max(e(A),e(B_3))}^{s(B_1)}$ has size at most $n_h - 1$. Moreover, since A and B_3 overlap, a minimal cover of $\mathcal{S}_h|_{t_s}^{\max(e(A),e(B_3))}$ has size at most two if $e(B_3) > e(A)$, and has size one otherwise. Together with the fact that a minimal cover of $\mathcal{S}_h|_{max(e(A),e(B_3))}^{e(B_2)}$ has size at most 2, by Proposition 5.9, therefore, $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{e(B_2)}) \leq n_h + 3$.

In the above we assumed that $e(B_3) < s(B_1)$. If $e(B_3) \geq s(B_1)$, then A and B_3 together constitute a cover of $\mathcal{S}_h|_{t_s}^{s(B_1)}$. So $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{e(B_2)}) \leq 4$. If no β-passage in $\mathcal{S}_h$ starts before $e(A)$, then by the above argument we can see that a minimal cover of $\mathcal{S}_h|_{t_s}^{e(A)}$ has size one and a minimal cover of $\mathcal{S}_h|_{e(A)}^{s(B_1)}$ has size at most $(n_h - 1)$. So $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{e(B_2)}) \leq n_h + 2$. Since $n_h \geq 1$, in any case, $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{e(B_2)}) \leq n_h + 3$.

Similarly, we can show that at most $(n_h - 1)$ γ-passages can overlap with $[e(B_2), t_f]$. Since no α- or β-passage can overlap with $[e(B_2), t_f]$, $\mathbf{dim}(\mathcal{S}_h|_{e(B_2)}^{t_f}) \leq n_h - 1$. By Proposition 5.9 and the above argument that $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{e(B_2)}) \leq n_h + 3$, we have that $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) \leq 2n_h + 2$.

We have considered the case that there exist two passages A and B such that: A is the passage, among the set of α-passages in $\mathcal{S}_h$ (i.e., set $\mathcal{A}$), with the largest end time, and B is a β-passage initiated no earlier than $e(A)$. If there is no β-passage, or all β-passages in $\mathcal{S}_h$ are initiated earlier than $e(A)$, then by a similar reasoning we can show that at most $(n_h - 1)$ γ-passages can overlap with $[t, t_f]$, where $t = max\{e(U) : U$ is an α- or β-passage in $\mathcal{S}_h\}$. Together with the fact that $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t}) \leq 2$, we have $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) \leq n_h + 1$.

If there is no α-passage (i.e., $\mathcal{A} = \emptyset$) but there is some β-passage in $\mathcal{S}_h$, then let $\mathcal{B}_1$ be the set of β-passages in $\mathcal{S}_h$ that are initiated at or before t_s, and $\mathcal{B}_2$ be the set of β-passages that are initiated after t_s. Note that the passages in $\mathcal{B}_2$ must overlap. Let $t_1 = \max\{e(U) \,|\, U \in \mathcal{B}_1\}$ if $\mathcal{B}_1 \neq \emptyset$, and $t_1 = t_s$ otherwise. Furthermore, let $t_2 = \min\{s(U) \,|\, U \in \mathcal{B}_2\}$ and $t_3 = \max\{e(U) \,|\, U \in \mathcal{B}_2\}$ if

$\mathcal{B}_2 \neq \emptyset$, and $t_2 = t_3 = t_1$ otherwise. Analogous to the above analysis, we can show that $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_1}) \leq 1$, $\mathbf{dim}(\mathcal{S}_h|_{t_1}^{t_2}) \leq n_h - 1$, $\mathbf{dim}(\mathcal{S}_h|_{t_2}^{t_3}) \leq 2$, and $\mathbf{dim}(\mathcal{S}_h|_{t_3}^{t_f}) \leq n_h - 1$. So $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) \leq 2n_h + 1$.

If there is no α- and β-passage in $\mathcal{S}_h$, then at most $(n_h - 1)$ γ-passages can overlap with $[t_s, t_f]$ (these passages must be resulted from some early α- and β-passages that occur before p_i completes its request). So $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) \leq n_h - 1$.

To summarize, $\mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) \leq 2n_h + 2$. Since $\mathbf{dim}(\mathcal{S}_h) = \mathbf{dim}(\mathcal{S}_h|_{t_{\min}}^{t_{\max}})$, where $t_{\min} = \min\{s(U) \,|\, U \in \mathcal{S}_h\}$ and $t_{\max} = \max\{e(U) \,|\, U \in \mathcal{S}_h\}$, by Proposition 5.9 $\mathbf{dim}(\mathcal{S}_h) \leq \mathbf{dim}(\mathcal{S}_h|_{t_{\min}}^{t_s}) + \mathbf{dim}(\mathcal{S}_h|_{t_s}^{t_f}) + \mathbf{dim}(\mathcal{S}_h|_{t_f}^{t_{\max}})$. Observe that some passages overlapping with $[t_s, t_f]$ may be initiated before t_s, and all passages overlapping with $[t_s, t_f]$ must be completed before t_f. So $\mathbf{dim}(\mathcal{S}_h|_{t_{\min}}^{t_s}) \leq 1$ and $\mathbf{dim}(\mathcal{S}_h|_{t_f}^{t_{\max}}) = 0$. Therefore, $\mathbf{dim}(\mathcal{S}_h) \leq 2n_h + 3$. □

Note that in the above lemma if $\mathbf{dim}(\mathcal{S}_h) = 2n_h + 3$ then there must be some ongoing α-passage when p_i completes its request for F_k. This means that the round of F_h must have already started when p_i completes its request. On the other hand, if the round starts after p_i has completed its request, then no α-passage can occur in this round (because any effective captain must have seen p_i's request when it is in line 9 of CTP-m, and so cannot set *turn* to F_h). As a result, there is no passage between $[t_s, s(B_1)]$ in Figure 5. So a minimal cover of $\mathcal{S}_h$ has size at most $n_h + 1$. We therefore have the following two corollaries.

Corollary 5.12 *Suppose a round of F_h starts before p_i completes a request for F_k, $\mathsf{F}_k \neq \mathsf{F}_h$. Let t_s be the time p_i completes the request, and let t_f be the time the round terminates. Moreover, let $\mathcal{S}_h$ be the set of passages that overlap with the interval $[t_s, t_f]$. Then*

$$\mathbf{dim}(\mathcal{S}_h) \leq 2n_h + 3$$

Corollary 5.13 *Suppose a round of F_h starts after p_i has completed a request for F_k, $\mathsf{F}_k \neq \mathsf{F}_h$. Let t_s be the time the round starts, and let t_f be the time it terminates. Moreover, let $\mathcal{S}_h$ be the set of passages that overlap with the interval $[t_s, t_f]$. Then*

$$\mathbf{dim}(\mathcal{S}_h) \leq n_h + 1$$

The above corollaries concern the case $\mathsf{F}_k \neq \mathsf{F}_h$. The case $\mathsf{F}_k = \mathsf{F}_h$ is considered below.

Lemma 5.14 *Suppose a round of F_k starts before p_i completes its request for F_k. Suppose further that p_i does not make a passage for its request in this round.*

Let t_s be the time p_i completes its request, and let t_f be the time this round of F_k terminates. Moreover, let $\mathcal{S}_k$ be the set of passages that overlap with the interval $[t_s, t_f]$. Then

$$\mathbf{dim}(\mathcal{S}_k) \leq n_k$$

Proof. If no non-γ-passage (i.e., α- or β-passages) overlaps with $[t_s, t_f]$, then at most $n_k - 1$ γ-passages can overlap with $[t_s, t_f]$ (as none of the γ-passages can be made by p_i). So $\mathbf{dim}(\mathcal{S}_k) = \mathbf{dim}(\mathcal{S}_k|_{t_s}^{t_f}) \leq n_k - 1$.

If some non-γ-passage made by, say p_j, overlaps with $[t_s, t_f]$, then if the passage starts after p_i has completed its request, then either p_j will capture p_i or p_i will have already entered CS before p_j attempts to capture it. Since by the lemma assumption p_i does not make a passage in this round, if there is any non-γ-passage overlapping with $[t_s, t_f]$, then it must be initiated before p_i completes its request. Let A be the non-γ-passage, among the non-γ-passages that overlap with $[t_s, t_f]$, with the largest end time. By the proof of Lemma 5.11 it can be seen that at most $(n_k - 2)$ γ-passages can overlap with $[e(A), t_f]$. (Note that none of these γ-passages and A can be made by p_i.) So $\mathbf{dim}(\mathcal{S}_k|_{e(A)}^{t_f}) \leq n_k - 2$. Since $\mathbf{dim}(\mathcal{S}_k) \leq \mathbf{dim}(\mathcal{S}_k|_{t_{\min}}^{e(A)}) + \mathbf{dim}(\mathcal{S}_k|_{e(A)}^{t_f})$, where $t_{\min} = \min\{s(U) \,|\, U \in \mathcal{S}_k\}$, $\mathbf{dim}(\mathcal{S}_k) \leq 2 + n_k - 2 = n_k$.

To summarize, in either case $\mathbf{dim}(\mathcal{S}_k) \leq n_k$. □

Theorem 5.15 (Time Complexity) *Let t_r be the time a philosopher p_i completes its request for F_k, and t_g be the time the request is granted (i.e., the time p_i initiates a passage through F_k). Moreover, let $\mathcal{S}$ be the set of passages that overlap with $[t_r, t_g]$ and that must be completed before p_i can initiate its passage. Then*

$$\mathbf{dim}(\mathcal{S}) \;\leq\; \max\,\{n_k,\; 2n_j + 3 : 1 \leq j \neq k \leq m\} \;+\; \left(\sum_{1 \leq j \leq m, j \neq k} n_j + 1\right)$$

Proof. By Lemma 5.7, p_i waits for at most m rounds of passages before a round of F_k is initiated in which it can make a passage through F_k. These include the round of passages that is already ongoing when p_i completes its request. By the proof, the $m-1$ rounds of passages that start after p_i has completed its request must all be different, and none of them is a round of F_k. So by Corollary 5.13, a minimal cover of the set of passages occurring in these $m-1$ rounds has size at most $\sum_{1 \leq j \leq m, j \neq k}(n_j + 1)$.

Consider the round of passages that is already ongoing when p_i completes its request. By Corollary 5.12 and Lemma 5.14, a minimal cover of the set of passages in $\mathcal{S}$ that occur in this round has size at most $\max\{n_k,\ 2n_j+3 : 1 \le j \ne k \le m\}$.

After these m rounds of passages, a round of F_k must follow, and by Case 2 of Lemma 5.4 p_i must make a passage through F_k in this round. Note that because the passages in this round are for the same forum, p_i needs not wait for any of them to complete in order to initiate its own passage. So by definition of $\mathcal{S}$ no passage in $\mathcal{S}$ comes from this round. Therefore,

$$\mathbf{dim}(\mathcal{S}) \ \le\ \max\left\{n_k,\ 2n_j+3 : 1 \le j \ne k \le m\right\} \ +\ \left(\sum_{1\le j\le m, j\ne k} n_j + 1\right)$$

□

Since $n_h \le n$, the time complexity is $O(m \cdot n)$. Note that in measuring $\mathbf{dim}(\mathcal{S})$ above we only consider passages that must be completed before p_i can initiate its passage; that is, we do not count those with which p_i's passage may proceed concurrently.

We now consider the concurrency of CTP-m, and recall that the degree of concurrency is measured by the maximum number of passages that can be initiated while a passage is ongoing and some philosopher is waiting for a different forum. Lemmas 5.10 and 5.11 imply that the degree of concurrency of CTP-m is at least $O(n_h^2)$. However, as shown below CTP-m can actually provide a degree of concurrency up to $O(n_h^3)$.

Theorem 5.16 (Concurrency) *Suppose p_j is in F_h and p_i is waiting for F_k, $k \ne h$. Then the number of passages that can still be initiated before p_j leaves F_h is at most*

$$\frac{n_h(n_h+1)(2n_h+1)}{12} + \frac{n_h(n_h-3)}{4}$$

Proof. Recall from the proof of Lemma 5.10 that among the $\frac{n_h(n_h+1)(2n_h+1)}{12} + \frac{3n_h(n_h+1)}{4}$ passages in $\mathcal{S}_h$, n_h of them are α-passages initiated before p_i completes its request, $\frac{n_h(n_h+1)(2n_h+1)}{12} + \frac{n_h(n_h-3)}{4}$ of them are β- and γ-passages initiated after p_i completes its request but before the last of the n_h α-passages terminates, and the rest $\frac{n_h(n_h+1)}{2}$ are β- and γ-passages initiated after all the α-passages terminate. So while the last α-passage is ongoing and p_i is waiting for F_k, at most $\frac{n_h(n_h+1)(2n_h+1)}{12} + \frac{n_h(n_h-3)}{4}$ more passages can be initiated. □

By comparing CTP-m with the simple centralized algorithm CTP-C presented in Section 3, we see that they have similar forum-switch complexity (m

for CTP-m, and $m+1$ for CTP-C); also both have time complexity $O(m \cdot n)$. However, CTP-C allows a virtually unbounded degree of concurrency (through the use of a centralized mechanism), while CTP-m can reach only $O(n^3)$ (in a fully distributed setting).

5.4 Remarks

We comment on some nontrivial design choices made for CTP-m. First, let us reconsider Lemma 5.10. In the absence of α-passages the size of $\mathcal{S}_h$ in Lemma 5.10 can be reduced to $n_h(n_h+1)/2$. In the algorithm α-passages occur because we allow a captain, upon seeing that no philosopher is interested in a different forum, to set *turn* to the same forum as its request. If we change line 9 of CTP-m so that a captain will always set *turn* to a different forum (as in the case of CTP-2), then α-passages are not possible. In this case, however, a minimal cover of $\mathcal{S}_h$ in Lemma 5.11 may still contain n_h+1 passages (2 overlapping β-passages and (n_h-1) γ-passages). So the time complexity of Theorem 5.15 is not affected by any order of magnitude. However, the degree of concurrency will then drop to $O(n_h^2)$. Therefore, in CTP-m we have opted for a higher degree of concurrency by allowing a philosopher to set *turn* to the same forum as its request.

Moreover, the order of execution of setting *turn* (line 9) and capturing philosophers (lines10-11) may also affect the time complexity. To see this, recall that in the proof of Lemma 5.11 all β-passages that are initiated no earlier than $e(A)$, i.e., the passages in $\mathcal{B}$, must overlap (at a common point). So it takes at most two passages to cover the passages in $\mathcal{B}$. If a philosopher sets *turn* after it captures philosophers, then the β-passages in $\mathcal{B}$ do not necessarily overlap (thereby increasing the time complexity). This is because a philosopher p_l may enter CS as a captain (assuming that p_l is interested in F_h and $turn = \mathsf{F}_h$), capture philosophers, and then find that some philosopher is interested in a different forum F_g. Before p_l sets *turn* to F_g, another p_j interested in F_h may have already read *turn* in line 7. Suppose p_l then sets *turn* to F_g and exits CS before p_j compares the value it has read (i.e., F_h) with the forum F_h it has requested (i.e., before p_j tests the predicate $turn = \mathsf{F}_h$ in line 7). Then when p_j finds that *turn* is (actually, was) F_h and enters CS, p_l has already left CS, and so their β-passages do not overlap.

A more significant boost to CTP-m's performance is by the extra clause "$turn = \mathsf{F}_k \vee all\_passive(turn)$" added to line 7 that a philosopher checks to see if it can enter CS. It is important to note that Lemma 5.6 would not hold if this clause is dropped from line 7. This is because a philosopher p_j interested in a forum F_h may have already proceeded to line 6 when a round of F_l starts. When the round of F_l terminates, if this clause is removed, then p_j may "sneak" into CS even if $turn \neq \mathsf{F}_h$ and some other philosopher is waiting for the forum specified by $turn$. As a result, the number of rounds of passages p_i needs to wait before it enters CS in Lemma 5.7 (i.e., the algorithm's forum-switch complexity) would be much more than m.

To illustrate, assume $m = 4$ and $turn = \mathsf{F}_0$. Consider the following scenario.[8]

1. p_3 requests F_3, finds that no one is interested in $\mathsf{F}_0, \mathsf{F}_1$, and F_2, and so it exits the while-loop and proceeds to line 6 (but has not yet executed line 6).
2. p_2 requests F_2, finds that no one is interested in F_0 and F_1, and so also proceeds to line 6.
3. p_1 requests F_1, finds that no one is interested in F_0, and so also proceeds to line 6.
4. p_0 requests F_0.
5. p_3 "sneaks" into CS (because $none\_in\_cs(\overline{\mathsf{F}_3}) \overrightarrow{\wedge}\ no\_successor(\overline{\mathsf{F}_3})$ evaluates to true). It then sets $turn$ to F_0 and exits CS. Then p_3 requests another entry to F_3.
6. p_2 "sneaks" into CS. It then sets $turn$ to F_3 and exits CS. Then p_2 requests another entry to F_2.
7. p_3 finds that $turn = \mathsf{F}_3$ and so it proceeds to enter CS. It then sets $turn$ to F_0, exits CS, and requests another entry to F_3.
8. p_1 "sneaks" into CS. It then sets $turn$ to F_2 and exits CS.
9. p_2 finds that $turn = \mathsf{F}_2$ and so it proceeds to enter CS. It then sets $turn$ to F_3 and exits CS.
10. p_3 finds that $turn = \mathsf{F}_3$ and so it proceeds to enter CS. It then sets $turn$ to F_0 and exits CS.
11. p_0 now finds that $turn = \mathsf{F}_0$ and so it proceeds to enter CS.

Therefore, before p_0 enters CS, the following 6 rounds of passages have bypassed: $\mathsf{F}_3^*, \mathsf{F}_2^*, \mathsf{F}_3, \mathsf{F}_1^*, \mathsf{F}_2, \mathsf{F}_3$, where F_i^* represents a round of F_i initiated by a philosopher that "sneaks" into CS. The scenario can be extended to $m = 5$ so that p_0 waits for the following rounds of passages before it enters CS to attend F_0:

$$\mathsf{F}_4^*, \mathsf{F}_3^*, \mathsf{F}_4, \mathsf{F}_2^*, \mathsf{F}_3, \mathsf{F}_4, \mathsf{F}_1^*, \mathsf{F}_2, \mathsf{F}_4^*, \mathsf{F}_3^*, \mathsf{F}_4$$

[8] Thanks to Wen-Jian Tsai for coming up with this scenario.

Note that p_4 can sneak into F_4 twice because the evaluation of $next\_op(turn)$ in line 5 requires an access to $turn$ and then an access to each philosopher's $flag$, and no particular ordering is assumed in accessing the $flag$s. As a result, after p_1 sneaks into CS to establish a round of F_1 and sets $turn$ to F_2 (because p_2 has requested another entry to F_2), p_4 can start to evaluate $next\_op(turn)$ in line 5. When p_4 learns that $turn = \mathsf{F}_2$, it may later find that no one is interested in F_3 and F_2 (because p_2 has already finished F_2), and so proceeds to line 6 waiting to sneak into CS. Similarly, p_3 can sneak into F_3 after p_1 has set $turn$ to F_2 because when p_3 reads $turn = \mathsf{F}_2$, it may later find that no philosopher is interested in F_2, and so obtains $next\_op(turn) = \mathsf{F}_3$ in line 5. So it can also proceed to line 6 waiting to sneak into CS.

Let a_k denote the number of rounds of passages p_0 may wait before it enters CS for the setting where the philosophers may like to hold k different fora. We leave the reader to show that the scenario can be generalized so that a_k satisfies the following recurrence relation:[9]

$$a_k = \begin{cases} a_{k-1} + a_{k-2} + 2 & k > 2 \\ 1 & k = 2 \\ 0 & k = 1 \end{cases}$$

Solving this recurrence relation we have

$$a_m = \frac{5+3\sqrt{5}}{10}\left(\frac{1+\sqrt{5}}{2}\right)^m + \frac{5-3\sqrt{5}}{10}\left(\frac{1-\sqrt{5}}{2}\right)^m - 2$$

Thus, an exponential number of rounds may pass before a philosopher's request is granted!

As mentioned earlier in Section 4.1, our algorithm is based on Knuth's algorithm for 2-process mutual exclusion [17]. When generalizing to n-process, Knuth's algorithm suffers an exponential number of overtakes: a process waits for $2^{n-1} - 1$ entries to CS before it enters CS. The exponential bound is reduced to linear by Eisenberg and McGuire [10] by properly assigning the $turn$ variable when a process exits CS. As illustrated above, CTP-m's forum-switch complexity (which corresponds to the above "overtakes" complexity when the Congenial Talking Philosophers problem is reduced to n-process mutual exclusion) could

[9] It helps to see the recurrence relation by renaming fora using the new index mapping: $\mathsf{F}_i \longrightarrow \mathsf{F}_{m-1-i}$. As a result, the turn is now assigned in a decreasing order $\mathsf{F}_i, \mathsf{F}_{i-1}, \mathsf{F}_{i-2}, \ldots$ instead of an increasing order $\mathsf{F}_i, \mathsf{F}_{i+1}, \mathsf{F}_{i+2}, \ldots$. Accordingly, in the above scenario for $m = 4$, p_0 now waits for the following 6 rounds of passages $\mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0, \mathsf{F}_2^*, \mathsf{F}_1, \mathsf{F}_0$ before it enters CS. For $m = 5$, the sequence becomes $\mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0, \mathsf{F}_2^*, \mathsf{F}_1, \mathsf{F}_0, \mathsf{F}_3^*, \mathsf{F}_2, \mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0$, and for $m = 6$, the sequence is $\mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0, \mathsf{F}_2^*, \mathsf{F}_1, \mathsf{F}_0, \mathsf{F}_3^*, \mathsf{F}_2, \mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0, \mathsf{F}_4^*, \mathsf{F}_3, \mathsf{F}_0^*, \mathsf{F}_1^*, \mathsf{F}_0, \mathsf{F}_2^*, \mathsf{F}_1, \mathsf{F}_0$.

also blow up to exponential if not properly designed. Unlike Eisenberg and McGuire's approach, we let a philosopher, prior to entering CS, check an additional condition to see if its forum is the most appropriate one to start.

6 Related Work and Conclusions

We have presented the Congenial Talking Philosophers problem to model group mutual exclusion in which resources can be shared by processes of the same group but the sharing cannot be done across groups. Although the problem occurs naturally in applications such as CSCW, to our knowledge, it has not been addressed in the literature thus far.

We have also presented an efficient and highly concurrent distributed algorithm CTP-m to solve the Congenial Talking Philosophers problem. In terms of forum-switch complexity, when a philosopher requests a forum, it waits for at most m rounds of passages before it attends the forum, where m is the total number of fora in the system. Within each round of passages, at most $O(n_h)$ passages suffice to cover all the passages that occur within the round, where n_h is the total number of philosophers that may potentially attend F_h. So the time complexity is $O(m \cdot n)$. In terms of concurrency, while a philosopher p_i occupies the meeting room and some other p_j is waiting for a different forum, CTP-m can admit $O(n^3)$ entries to the meeting room to join the ongoing forum with p_i.

For comparison, we have presented two algorithms, one centralized and the other semi-distributed, for the Congenial Talking Philosophers problem. Both algorithms are able to claim a virtually unbounded degree of concurrency by using a centralized mechanism to monitor philosophers' states. The centralized mechanism, however, also makes them more vulnerable to faults. In particular, the semi-distributed algorithm may result in unbounded time and forum-switch complexity. Even for the centralized algorithm, its time and forum-switch complexity is approximately the same as our distributed solution.

As discussed in Section 2, the Congenial Talking Philosophers problem is more general than the conventional n-process mutual exclusion and the Readers and Writers problems. Our algorithm CTP-m also offers an appealing solution for these problems. For n-process mutual exclusion, a process waits for at most n passages before it enters the critical section. Note that this includes the one that is already ongoing when the process makes its request for the critical section. So, after a process requests the critical section, at most $n-1$ entries to the critical section may proceed before the process, which is obviously a lower bound for the mutual exclusion problem. (For a survey of mutual exclusion algorithms see [24, 4, 25, 20].) For the Readers and Writers problem, a *wait-free* approach is usually adopted within the realm of shared memory to allow concurrent reading while writing [23, 13]. In this approach, $n+1$ extra copies of the shared object are used to allow the readers to keep track of the most recent version of the shared object. CTP-m, on the other hand, allows concurrent reading without introducing extra copies of the shared object, but it does not allow concurrent reading while writing.

A generalization of n-process mutual exclusion that allows at most l processes to be in the critical section simultaneously (known as the *l-exclusion* problem) has been proposed by Fisher, et al. [12], and subsequently studied by Afek, et al. [1]. However, there is no direct connection between the l-exclusion problem and the Congenial Talking Philosophers problem in the sense that the solution for one problem cannot be straightforwardly applied to the other.

In light of the l-exclusion problem, Congenial Talking Philosophers can be further generalized to model "l-forum exclusion", where there are l meeting rooms for the philosophers and so at most l fora can be in session simultaneously. This new problem can be applied in situations in which a resource can be shared by processes of the same group but not by processes of different groups, and l copies of the resource are available. This generalized problem can be easily reduced to the three fundamental problems: n-process mutual exclusion, Readers and Writers, and l-exclusion, but not vice versa. It is therefore interesting to see how this more general problem can be solved efficiently and in a distributed manner. Other future work includes studying various bounds of the Congenial Talking Philosophers problem, such as time, concurrency, and the number of variables required.

Acknowledgments. I would like to thank Wen-Jian Tsai for providing some initial thoughts on the algorithms. He also provided many useful comments during the preparation of the manuscript. I would also like to thank Jen-Yi Liao for coining the term "Congenial Talking Philosophers" for our problem. His original suggestion was "Congenial Dining Philosophers" for the dining problem where there is only one dining table, and philosophers interested in the same style of food, e.g., Chinese, French, or Italian, can come to the table to share the food. Last, but not least, I would like to thank Mark Moir and the anonymous referees for their valuable comments, and Idit Keidar and Paul Attie for proofreading the paper.

References

[1] Yehuda Afek, D. Dolev, Eli Gafni, M. Merritt, and N. Shavit. A bounded first-in, first-enabled solution to the ℓ-exclusion problem. *ACM Transactions on Programming Languages and Systems*, 16(3):939–953, May 1994.

[2] Divyakant Agrawal, Amr El Abbadi, and A. E. Lang. The performance of protocols based on locks with ordered sharing. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):805–818, October 1994.

[3] James H. Anderson and Mark Moir. Universal constructions for large objects. *IEEE Transactions on Parallel and Distributed Systems*, 2000. To appear.

[4] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* Englewood Cliffs NJ: Prentice-Hall, 1990.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database.* Addison-Wesley, 1987.

[6] James E. Burns. Mutual exclusion with linear waiting using binary shared variables. *ACM SIGACT News*, 10(2):42–47, summer 1978.

[7] K. Mani Chandy and Jayadev Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, October 1984.

[8] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation,* Chapter 14: Committee Coordination. Addison-Wesley, 1988.

[9] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with readers and writers. *Communications of the ACM*, 14(10):667–668, October 1971.

[10] M. A. Eisenberg and M. R. McGuire. Further comments on Dijkstra's concurrent programming control problem. *Communications of the ACM*, 15(11):999, November 1972.

[11] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a data base system. *Communications of the ACM*, 19(11):624–633, November 1976.

[12] Michael J. Fischer, Nancy A. Lynch, James E. Burns, and Allan Borodin. Resource allocation with immunity to limited process failure (preliminary report). In *20th Annual Symposium on Foundations of Computer Science*, pages 234–254, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

[13] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

[14] Maurice Herlihy. A methodology for implementing highly concurrent objects. *ACM Transactions on Programming Languages and Systems*, 15(5):745–770, November 1993.

[15] Yuh-Jzer Joung. Asynchronous group mutual exclusion. Technical report, Department of Information Management, National Taiwan University, Taipei, Taiwan, 1998.

[16] Yuh-Jzer Joung. The congenial talking philosophers problem in computer networks (extended abstract). In *Proceedings of the 13th International Symposium on DIStributed Computing (DISC99)*, Lecture Notes in Computer Science 1693, pages 195–209. Springer, 1999.

[17] D. E. Knuth. Additional comments on a problem in concurrent programming control. *Communications of the ACM*, 9(5):321–322, May 1966.

[18] Henry F. Korth. Locking primitives in a database system. *Journal of the ACM*, 30(1):55–79, January 1983.

[19] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[20] Nancy A. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, 1996.

[21] Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, October 1979.

[22] G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.

[23] G. L. Peterson. Concurrent reading while writing. *ACM Transactions on Programming Languages and Systems*, 5(1):46–55, January 1983.

[24] Michel Raynal. *Algorithms for Mutual Exclusion*. MIT Press, Cambridge, MA, 1986.

[25] A. Silberschatz and P. Galvin. *Operating System Concepts.* Addison-Wesley, fourth edition, 1994.

[26] William E. Weihl. Local atomicity properties: Modular concurrency control for abstract data types. *ACM Transactions on Programming Languages and Systems*, 11(2):249–283, April 1989.