On the generalized dining philosophers problem*

Oltea Mihaela Herescu   Catuscia Palamidessi

Department of Computer Science and Engineering, The Pennsylvania State University
Pond Lab, University Park, PA 16802-6106 USA

{herescu,catuscia}@cse.psu.edu

ABSTRACT

We consider a generalization of the dining philosophers problem to arbitrary connection topologies. We focus on symmetric, fully distributed systems, and we address the problem of guaranteeing progress and lockout-freedom, even in presence of adversary schedulers, by using randomized algorithms. We show that the well-known algorithms of Lehmann and Rabin do not work in the generalized case, and we propose an alternative algorithm based on the idea of letting the philosophers assign a random priority to their adjacent forks.

Categories and Subject Descriptors

D.4.1 [**Software**]: Operating Systems—*Process Management*; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

1. INTRODUCTION

The problem of the dining philosophers, proposed by Dijkstra in [1], is a very popular example of control problem in distributed systems, and has become a typical benchmark for testing the expressiveness of concurrent languages and of resource allocation strategies.

The typical dining philosophers sit at a round table in positions alternated with forks, so that there is a fork between each two philosophers, and a philosopher between each two forks. Each philosopher can pick up only the forks immediately to his right and to his left, one at the time, and needs both of them to eat. The aim is to make sure that if there are hungry philosophers then some of them will eventually eat (*progress*), or, more ambitiously, that every hungry philosopher will eventually eat (*lockout-freedom*).

The solutions to the problem of the dining philosophers depend fundamentally on the assumptions made on the system. If we do not impose an initial symmetry, or do not impose that the system be completely distributed, then several solutions are possible. Some examples are:

- The forks are ordered and each philosopher tries to get first the adjacent fork which is higher in the ordering.
- The philosophers are colored yellow and blue alternateadly. The yellow philosophers try to get first the fork to their left. The blue ones try to get first the fork to their right.
- There is a central monitor which controls the assignment of the forks to the philosophers.
- There is a box with $n-1$ tickets, where n is the number of the philosophers, and each philosopher must get a ticket before trying to get the forks.

In the first two solutions above, the system is not symmetric. In the last two, it is not fully distributed.

Of course, the problem becomes much more challenging when we impose the conditions of symmetry and full distribution. More precisely, symmetry means that the philosophers are indistinguishable, as well as the forks. The philosophers run the same program, and both the forks and the philosophers are all in the same initial state. Full distribution means that there are no other processes except the philosophers, there is no central memory, all philosophers run independently, and the only possible interaction is via a shared fork.

The conditions of symmetry and full distribution are interesting also for practical considerations: in several cases it is desirable to consider systems which are made of copies of the same components, and have no central control or shared memory. In particular, symmetry offers advantages at the level of reasoning about the system, as it allows a greater modularity, and at the level of implementation of concurrent languages, as it allows a compositional compilation. Full distribution is usually convenient as it avoids the overhead of a centralized control.

Lehmann and Rabin have shown in [9] the remarkable result that there are no *deterministic* solutions to the dining philosophers problem, if symmetry and full distributions are imposed, and if no assumption (except fairness) are made on the scheduler. The only possible solution, in such conditions, are *randomized* algorithms, that allow to eventually

*Work supported by the NSF-POWRE grant EIA-0074909.

break the initial symmetry with probability 1. In [9] two such algorithms are proposed, the first guarantees progress, the second guarantees also lockout-freedom.

There are two proofs of correctness of the Lehmann and Rabin algorithms, one in [9] and another one, more structured and formal, in [4, 10]. They both depend in an essential way on the topology. In particular, they depend on the fact that one fork can only be shared by two philosophers (cfr. Lemma 1 in [9], Lemma 7.13 in [4], and Lemma 6.3.14 in [10]). Therefore, a question naturally arises: Would the solutions of Lehmann and Rabin still work in the case of more general connection structures? The problem is also of practical relevance, since the kind of resource network represented by the classic formulation is very restricted. In this paper we investigate this question and show that the answer is no: In most situations, both the algorithms of Lehmann and Rabin fail. We then propose another solution, still randomized but based on a rather different idea: we let each philosopher try to establish a partial order on forks, by assigning a random number to his adjacent forks. In other words, we use randomization for breaking the initial symmetry and achieving a situation in which the forks are partially ordered. Finally, we propose a variant of the algorithm which ensures that no philosopher will starve (lockout-freedom). The algorithms are robust wrt every fair scheduler.

Our motivation for this work comes from the project of providing a distributed implementation for the π-calculus [5]. So far, only the so-called asynchronous subset has been implemented [8]. In [7], the second author has shown that the full π-calculus is strictly more expressive than its asynchronous subset, and, more in general, that there is no hope of implementing the π-calculus with deterministic methods. In [6] Nestmann has shown that the gap in expressive power, and the difficulty in the implementation with deterministic methods, is due to the mixed (input and output) guarded choice construct of the π-calculus. Such mechanism, however, would be very desirable as it provides a powerful programming primitive for solving distributed conflicts. Thus, we are considering a randomized implementation. We have developed an asynchronous probabilistic π-calculus (π_{pa}, [3]), and we are currently investigating a translation from π to π_{pa} that requires solving a resource allocation problem similar to the one of the generalized dining philosophers (the resources corresponds the channels of the π-calculus). The restriction to symmetric solutions comes from the desire of providing a *compositional* translation. To our knowledge, there has been only a previous proposal for a symmetric and fully distributed implementation of a concurrent language with guarded choice ([2]), but such proposal works only under the assumption of "good" schedulers, i.e. a scheduler which behaves uniformly through the computation regardless of the actions performed by the processes.

2. THE GENERALIZED DINING PHILOSOPHERS PROBLEM

In this section we introduce a generalization of the dining philosophers problem. The generalization consists in relaxing the assumptions about the topology of the system. In the classic problem the philosophers and the forks are distributed along a ring (table) in alternated position. On the contrary, we consider arbitrary connection topologies, and in particular we admit the possibility that a fork is shared by more than two philosophers. Thus the number of forks and the number of philosophers is not necessarily the same. The only constraint we impose on the topology is that each philosopher is connected (has access) to two distinct forks. For the rest, the new formulation coincides with the classic one.

DEFINITION 1. *A generalized dining philosopher system consists of $n \geq 1$ philosophers and $k \geq 2$ forks. Unlike the classic case, n and k may be different numbers, and a fork can be shared by an arbitrary (positive) number of philosophers. Like in the classic case, every philosopher has access to two forks, which he will refer to as left and right. Every philosopher can think or eat. When a philosopher wants to eat, he must pick up the two forks. He can pick up only one fork at the time. He cannot pick up a fork if his neighbor is already holding it. He cannot eat forever. After eating the philosopher releases the two forks and resumes thinking.*

Figure 1 shows some examples of generalized dining philosopher systems. Here and in the rest of the paper, we represent a system as an undirected graph where the nodes are the forks (represented by sticks in Figure 1), and the arcs are the philosophers (represented by circles in Figure 1). Obviously, the forks accessible to a philosopher are the adjacent nodes. Note that we adopt the more general definition of graph, which allows the presence of more than one arc between two nodes (some textbooks use the term *multigraph*).

The goal is the same as in the classic problem: to program the philosophers so that hungry philosophers will eventually eat. Following the standard terminology, we will say that a solution ensures progress (wrt a set of philosophers) if it guarantees that, whenever a philosopher of the set is hungry, then a philosopher of the same set (not necessarily the same philosopher) will eventually eat[1]. A solution is *lockout-free* (wrt a set of philosophers) if it guarantees that, whenever a philosopher of the set is hungry, then the same philosopher will eventually eat.

We shall consider only *fully distributed* and *symmetric solutions*, namely algorithms where the only processes are the philosophers, the only shared variables are the forks, all philosophers run identical programs and both the philosophers and the forks are in the same initial state. We assume that *test-and-set* operations on the forks are performed atomically.

A computation consists in an interleaving of actions performed by the philosophers. Such interleaving is controlled by an *adversary* (or *scheduler*). We assume that the adversary has complete information of the past of the computation, and can decide its next step on the basis of that information. We consider only *fair* adversaries, namely adversaries that ensure that each philosopher executes infinitely many actions in each of the possible computations.

We will consider *randomized algorithms*, namely algorithms which allow a philosopher to select randomly between two

[1] In [9] this property is called *deadlock-freedom*.

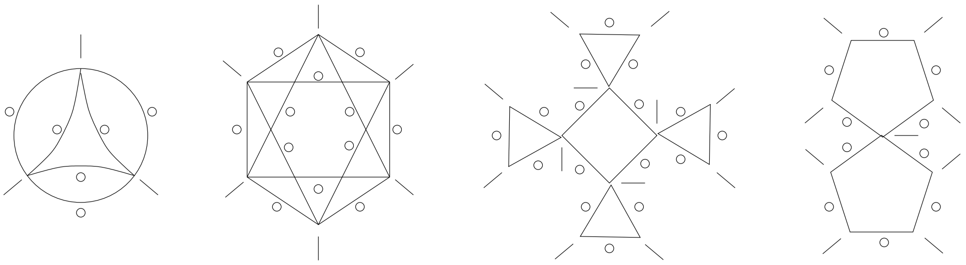


Figure 1: Some examples of generalized dining philosophers. From left to right: 6 philosophers, 3 forks. 12 philosophers, 6 forks. 16 philosophers, 12 forks. 10 philosophers, 9 forks.

1. *think*;
2. *fork* := *random_choice*(*left*, *right*);
3. *if isFree*(*fork*) *then take*(*fork*) *else goto* 3;
4. *if isFree*(*other*(*fork*))
   then take(*other*(*fork*))
   else {*release*(*fork*); *goto* 2}
5. *eat*;
6. *release*(*fork*); *release*(*other*(*fork*));
7. *goto* 1;

Table 1: The algorithm LR1.

or more alternatives. The outcome of the random choice depends on a probability distribution, and it is not controlled by the adversary. For this reason, even under the same adversary, different computations may be possible. This model of computation has been formalized by Lynch and Segala by introducing the concept of *probabilistic automata* [11, 10].

3. LIMITATIONS OF THE ALGORITHMS OF LEHMANN AND RABIN

In this section we show that the randomized algorithms of Lehmann and Rabin presented in [9] do not work anymore in the general case.

We start by recalling the first algorithm of Lehmann and Rabin, LR1 for short. Each philosopher runs the code written in Table 1.

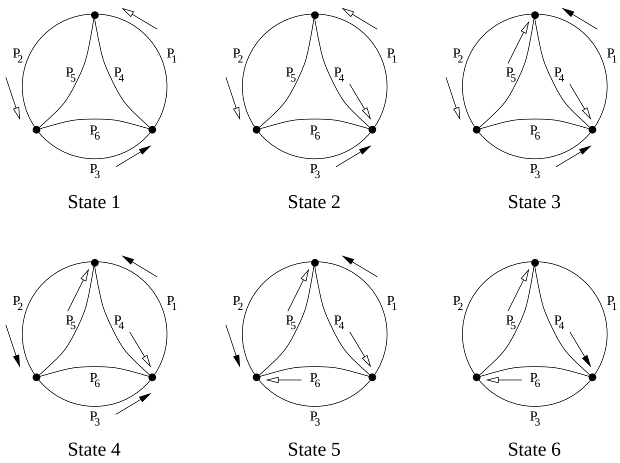
Following standard conventions we assume that the action *think* may not terminate, while all the other ones are supposed to terminate. The test-and-set operations on the forks, in Steps 3 and 4, are supposed to be executed atomically. Each outcome (left or right) of the random draw has a positive probability and the sum of the probabilities is 1. In the classic algorithm the probability is evenly distributed (1/2 for left and 1/2 for right). However our negative results do not depend on this assumption.

It has been shown in [9] that, for the classic dining philosophers, LR1 ensures progress with probability 1 under every fair scheduler. A more formal proof of this result can be found in [4, 10].

In the generalized case this result does not hold anymore. Let us illustrate the situation with an example. We use the following notation: An empty arrow, associated with a philosopher and pointing towards a fork, denotes that the philosopher has committed to that fork (has selected that fork with the random choice instruction) but he has not taken it yet. A filled arrow denotes that the philosopher is holding the fork, namely he has taken the fork and has not released it yet. From now on, we will represent the nodes (forks) as bullets, instead than as sticks.

For the sake of simplicity, for the moment we relax the fairness requirement. We will discuss later how to make the example valid also in presence of fairness.

Consider the system on the leftmost side of Figure 1, and consider State 1 depicted in the figure below. Clearly, this state is reachable from the initial state (where all philosophers are at the beginning of the program, i.e. thinking) with a non-null probability.



The scheduler chooses P_4 next. If P_4 commits to the free fork, then he will take it and then try to get the other fork. Since the other fork is taken by P_3, P_4 has to release the first fork and draw again. The scheduler keeps selecting P_4 until he commits to the fork taken by P_3. When this commitment occurs we are in State 2.

Next the scheduler selects P_1 and P_1 takes the fork he had committed to. Then the scheduler keeps scheduling P_5 until P_5 commits to the fork taken by P_1 (like it was done for P_4). This is State 3.

Then the scheduler selects P_2, and P_2 takes the fork he had committed to. This situation is represented by State 4.

The scheduler continues with P_3. P_3 finds his second fork taken by P_2 and therefore releases the fork that he currently controls. P_6 is then scheduled, until it commits to the fork taken by P_2. This is State 5.

Finally the scheduler runs P_2, and P_2 will have to release his fork, since the other fork is taken by P_1. Then P_4 is selected, and he takes the fork he had committed to. Then the scheduler selects P_1, which will have to release his fork since the other one is taken by P_4. This is State 6.

Observe now that State 6 is isomorphic to State 1, in the sense that they differ only for the names of the philosophers. The scheduler can then go back to State 1 and then repeat these steps forever, thus inducing a computation in which no philosopher is able to eat. Note that the probability of a computation of this kind is $1/4$, which is the probability of reaching a state isomorphic to State 1 already at the first attempt. (We are assuming that the probability of picking a particular adjacent fork is evenly distributed between left and right, i.e. it's $1/2$. If this is not the case then the figure above will be different from $1/4$, but it will still be positive.) It's easy to see that, by repeating the attempt to reach State 1 (possibly after some philosopher has eaten), the scheduler can *eventually* induce a cycle like the above one with probability 1.

Unfortunately the scheduler considered in this example is unfair. In fact, it keeps selecting one philosopher (for instance P_4) until it commits to a taken fork. If the philosopher chooses forever the free fork, then the resulting computation is unfair. Although such a computation has probability 0, according to the definition of fairness, the scheduler is unfair.

However, it is easy to modify the scheduler so to obtain a fair scheduler which achieves the same result, namely a no-progress computation with non-null probability, although smaller than $1/4$. Consider a variant of the above scheduler which keeps selecting a "stubborn philosopher" for a finite number of times only, but which increases this number at every round. By "round" here we mean the computation fragment which goes from State 1 to State 6, and back to State 1, as described above. Let n_k be the maximum number of times which the scheduler is allowed to select the same philosopher during the k-th round. Choose n_k to be big enough so that the probability that the scheduler actually succeeds to complete the k-th round is $1 - p^k$ with $p \leq 1/2$. Consider an infinite computation made of successive successful rounds. The probability of this computation is greater than or equal to

$$\frac{1}{4}\, \Pi_{k=1}^{\infty}(1 - p^k).$$

It is easy to prove by induction that for every $m \geq 1$,

$$\Pi_{k=1}^{m}(1 - p^k) \geq 1 - p - p^2 + p^{m+1}$$

holds. Hence we have

$$\Pi_{k=1}^{\infty}(1 - p^k) \geq 1 - p - p^2.$$

Furthermore, by the assumption $p \leq 1/2$, we have

$$1 - p - p^2 \geq 1/4.$$

3.1 A general limitation to the first algorithm of Lehmann and Rabin

We have seen that there is at least one example of graph in which LR1 does not work. One could hope that this example represents a very special situation, and that under some suitable conditions LR1 could still work in more general cases than just the standard one. Unfortunately this is not true: It turns out that as soon as we allow one fork of the ring to be shared by an additional philosopher, LR1 fails.

In the following, we will call *ring* (or *cycle*) a graph which has k nodes, say $0, 1, \ldots k-1$, and k arcs connecting the pairs $(0,1), (1,2), \ldots, (k-1, 0)$.

THEOREM 1. *Consider a graph G containing a ring subgraph H, and such that one of the nodes of H has at least three incident arcs (i.e. an additional arc in G besides the two in H). Then it is possible to define a fair scheduler for LR1 such that the probability of a computation in which the arcs (philosophers) in H make no progress is strictly positive.*

Proof (Sketch) Figure 2 represents the subgraph of G consisting of

- the ring H (a hexagon in the figure, but the number of vertices is not important) with a node f having (at least) three incident arcs,
- the arc P in G but not in H which is incident on f, and
- the node g adjacent to f via P.

It does not matter whether g is a node in H or not.

Figure 2 shows a possible sequence of states induced by a scheduler S. State 1 is reachable from the initial state (where all philosophers are thinking) with a non-null probability. The scheduler controls the directionality of the arrows by means of the technique explained in the example in previous section. The state transitions should be rather clear, except maybe for the last one. That transition (between State 5 and State 6) is achieved by the following sequence of actions:

- schedule P and let him eat (this is always possible - the scheduler can always make g free at the moment P needs it)

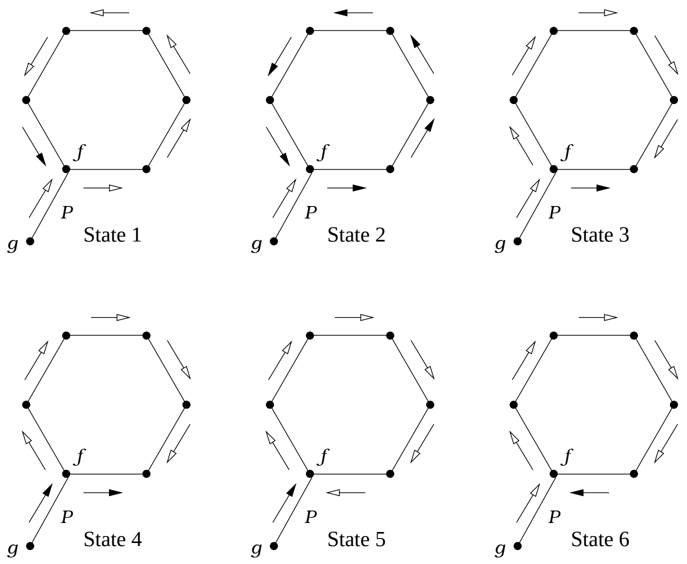


Figure 2: A winning scheduling strategy against the algorithm LR1.

- schedule the philosopher adjacent to P which is committed to f, and let him take f
- keep scheduling P until he commits to f.

State 6 is symmetric to State 1 and we can therefore define an infinite computation, where no philosopher in H eats, by repeating the actions which bring from State 1 to State 6, and then back to State 1, and so on.

Again, the scheduler S illustrated here is not fair, but we can obtain a fair scheduler S' which approximates S by letting the "level of stubbornness" of S' increase at each round, following the technique used in the example above. □

3.2 The second algorithm of Lehmann and Rabin

In this section we consider the second algorithm of Lehmann and Rabin, presented in [9] as a lockout-free solution to the classic dining philosophers.

We consider here a slight generalization of the original algorithm suitable for the a generic topology. Hereafter we will refer to it as LR2. We assume that each fork is provided with the following data structures:

- A list of incoming requests r, with operations *isEmpty*, *insert*, and *remove*. Initially the list is empty.
- A "guest book" g, namely a list which keeps track of the philosophers who have used the fork.

The idea is that when a philosopher gets hungry, he inserts his name *id* in the request list of the adjacent forks[2]. After the philosopher has eaten, he removes his name from these lists, and signs up the guest books of the forks. Before picking up a fork, a philosopher must check that there are no other incoming requests for that fork, or that the other philosophers requesting the fork have used it after he did. This condition will be represented, in the algorithm, by the condition *Cond(fork)*[3].

Table 2 shows the code run by each philosopher.

The negative result expressed in Theorem 1 does not hold for LR2. In fact, once P has eaten, he cannot take Fork f before the neighbor has eaten as well. However the class of graphs in which LR2 does not work is still fairly general:

THEOREM 2. *Consider a graph G containing a ring subgraph H, and such that two of the nodes in H are connected at least by three different paths (i.e. an additional path P in G besides the two in H). Then it is possible to define a fair scheduler for LR2 such that the probability of a computation in which the arcs (philosophers) of H and P make no progress is strictly positive.*

Proof (Sketch) The proof is illustrated in Figure 3, which shows the part of G containing the ring H and the additional path between two nodes of H.

Like before, the computation illustrated in the figure is induced by an unfair scheduler S, but by following the usual

[2] We do not need to assume that all philosophers have different *ids*, but simply that those who share a fork are distinguished form each other. This assumption does not violate the symmetry requirement. In fact, the distinction between the adjacent philosophers could be stored in the fork and used only within the operations on the fork.

[3] In the original algorithm the list of incoming request is replaced by two switches associated to the two adjacent philosophers, and instead of the guest book there is a simple variable which indicates which one of the adjacent philosophers has eaten last.

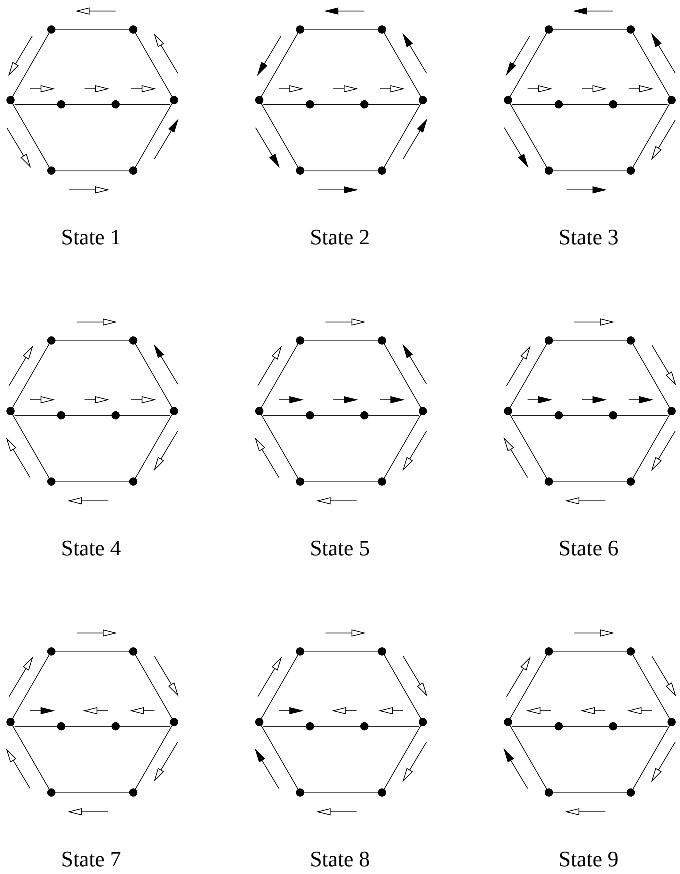


Figure 3: A winning scheduling strategy against the algorithms LR1 and LR2.

```
1.  think;
2.  insert(id, left.r);  insert(id, right.r);
3.  fork := random_choice(left, right);
4.  if isFree(fork) and Cond(fork)
        then take(fork)
        else goto 4;
5.  if isFree(other(fork))
        then take(other(fork))
        else {release(fork);  goto 3; }
6.  eat;
7.  remove(id, left.r);  remove(id, right.r);
8.  insert(id, left.g);  insert(id, right.g);
9.  release(fork);  release(other(fork));
10. goto 1;
```

Table 2: The algorithm LR2.

technique we can define a fair scheduler S' which approximates S and which achieves the same result. Note that none of the philosophers in H and in the additional path ever gets to eat, hence the modification of LR2 wrt LR1, namely the test *Cond*(*fork*), is useless: *fork*.g remains forever empty. $\square$

4. A DEADLOCK-FREE SOLUTION

In this section we propose a symmetric and fully distributed solution to the generalized dining philosophers and show that it makes progress with probability 1.

Our algorithm works as follows. Each fork has associated a field *nr* which contains an integer number ranging in the interval $[0, m]$, with $m \geq k$, where k is the total number of forks in the system. Initially *nr* is 0 for all the forks. Each philosopher can change the *nr* value of a fork when he gets hold of it, and he tries to make sure that the *nr* values of its adjacent forks become and are maintained different. In order to ensure that this situation will be eventually achieved, each new *nr* value is chosen *randomly*. Note that this random choice is necessary to break the symmetry, otherwise, in presence of a ring, a malicious scheduler could induce a situation where one philosopher changes one fork, then his neighbor changes the other fork to the same value, and so on, for all the forks in the ring.

Our algorithm is similar to LR1, except that the choice of

```
1. think;
2. if left.nr > right.nr
       then fork := left
       else fork := right;
3. if isFree(fork) then take(fork) else goto 3;
4. if fork.nr = other(fork).nr
       then fork := random[1, m];
5. if isFree(other(fork))
       then take(other(fork))
       else {release(fork); goto 2}
6. eat;
7. release(fork); release(other(fork));
8. goto 1;
```

Table 3: The algorithm GDP1.

the first fork is done by picking the one with the highest *nr* value (if they are different), instead than randomly. The other difference is that, as explained before, the philosopher may change the *nr* value of a fork when it finds that it is equal to the *nr* value of the other fork. This is done by calling $random[1, m]$, which returns a natural number in the interval $[1, m]$, selected probabilistically. We assume for simplicity that the probability of the outcome is evenly distributed among the numbers in the interval. The algorithm is illustrated in Table 3. We will refer to it as GDP1.

We prove now that GDP1 makes progress, under every fair scheduler, with probability 1. The proof is formalized in terms of the *progress* and *unless* statements introduced in [4, 10]. A progress statement is denoted by $S \xrightarrow[p]{A} S'$, where S and S' are sets of states, p is a probability, and A is a class of adversaries. Its meaning is that starting from any state in S, under any adversary in A, a state in S' is reached with probability at least p. An unless statement is of the form S *unless* S' and means that, if the system is in one of the states of S, then it remains in S (possibly moving through different states of S) until it reaches a state in S'.

Let us define T to be the set of states in which some philosopher tries to eat (*trying section*, steps 2 through 5), and E to be the set of states in which some philosopher is eating. We will show that $T \xrightarrow[1]{F} E$, where F is the class of all fair adversaries. The following properties of progress statement, proved in [4, 10], will be useful for our purposes.

LEMMA 1 (CONCATENATION). *If* $S \xrightarrow[p_1]{A} S'$ *and* $S' \xrightarrow[p']{A} S''$, *then* $S \xrightarrow[pp']{A} S''$.

LEMMA 2 (UNION). *If* $S_1 \xrightarrow[p_1]{A} S_1'$ *and* $S_2 \xrightarrow[p_2]{A} S_2'$, *then* $S_1 \cup S_2 \xrightarrow[p]{A} S_1' \cup S_2'$ *with* $p = min\{p_1, p_2\}$.

LEMMA 3 (PERSISTENCE WINS). *If* $S \xrightarrow[p]{F} S'$ *with* $p > 0$, *and* S *unless* S', *then* $S \xrightarrow[1]{F} S'$.

We are now ready to prove the correctness of GDP1.

THEOREM 3. $T \xrightarrow[1]{F} E$.

Proof Let us denote by C_1 the set of states in which there is one cycle in the graph where all adjacent forks have different numbers. C_2 is the set of states in which there are two cycles in the graph where all adjacent forks have different numbers, and so on.

We have the following progress statements:

- $T \xrightarrow[p]{F} (T \cap C_1) \cup E$, with $p \geq m!/m^k(m-k)!$. In fact, one of the trying philosophers, say P_1, will find the first fork free and will pick it up. Then, either he will find also his second fork free, and therefore will eat, or it will find the second fork taken by another philosopher, say P_2. Again, either P_2 will eat, or will find his second fork taken by another philosopher, say P_3, etc. Since the number of philosophers is finite, we will end up either with one philosopher eating, or with a ring of forks all picked up as first forks at least once. Since each philosopher changes the *nr* value of the first fork if this value is equal to that of the other fork, the adjacent forks of this ring will get all different values with probability p not smaller than $m!/m^k(m-k)!$ (this is the probability that, if we assign randomly values in the range $[1, m]$ to the nodes of a complete graph of cardinality k, all the nodes get a different value). Note that, by the assumption $m \geq k$, we have $p > 0$.
- $T \cap C_1 \xrightarrow[p]{F} (T \cap C_2) \cup E$. Similar to previous point.
- ...
- $T \cap C_{h-1} \xrightarrow[p]{F} (T \cap C_h) \cup E$. Similar to previous point.
- $T \cap C_h \xrightarrow[1]{F} E$. When all possible cycles in the graph have adjacent nodes with different *nr* values, then the algorithm works like a hierarchical resource allocation algorithm based on a partial ordering: Let P be the first philosopher who is holding the first fork, and such that the *nr* value of the other fork f is the smallest of all the forks adjacent to f. Then either P or one of his neighbors will eat.

From the above statements, and by using Lemma 1, the obvious fact that $E \xrightarrow[1]{F} E$, and Lemma 2, we derive

$$T \xrightarrow[p^h]{F} E.$$

```
1.  think;
2.  insert(id, left.r); insert(id, right.r);
3.  if left.nr > right.nr
        then fork := left
        else fork := right;
4.  if isFree(fork) then take(fork) else goto 4;
5.  if fork.nr = other(fork).nr
        then fork := random[1, m];
6.  if isFree(other(fork))
        then take(other(fork))
        else {release(fork); goto 3}
7.  eat;
8.  remove(id, left.r); remove(id, right.r);
9.  insert(id, left.g); insert(id, right.g);
10. release(fork); release(other(fork));
11. goto 1;
```

Table 4: The algorithm GDP2.

On the other hand, it's clear that, since philosophers keep trying until they eat, we have also

$$T \textit{ unless } E.$$

Therefore, by applying Lemma 3, we conclude $T \xrightarrow[1]{F} E$. $\square$

Note that GDP1 does not guarantee that we will reach, with probability 1, a situation where all adjacent forks will have a different *nr*. Not even if all philosophers are in the trying section infinitely often. This is because some philosophers may never succeed to pick up a fork, for instance because they are always scheduled when their neighbors are eating.

5. A LOCKOUT-FREE SOLUTION

The algorithm GDP1 presented in the previous section is not lockout-free. In fact, consider two adjacent philosophers, P_1 and P_2, which share a fork f with a *nr* value which is smaller than the value of the other fork g of P_1. Then P_1 will keep selecting g as first fork, and the scheduler could keep scheduling the attempt of P_1 to pick the second fork, f, only when f is held by P_2.

We now propose a lockout-free variant of GDP1. The idea is to associate to each fork a list of incoming requests r, and a "guest book" g, like it was done in Section 3.2. The test *Cond*(*fork*) is defined in the same way as in Section 3.2. The new algorithm, that we will call GDP2, is shown in Table 4.

We show now that GDP2 is lockout-free. In the following, T_i will represent the set of states in which the philosopher P_i is trying to eat, and E_i the situation in which the philosopher P_i is eating.

THEOREM 4. $T_i \xrightarrow[1]{F} E_i$.

Proof Let us denote by $C_{i,r}$ is the set of states in which there are r cycles containing the arc P_i, and where all adjacent forks have different numbers. Furthermore, let us use $W_{i,s}$ to represent the set of states in which there are s philosophers connected to P_i which have already eaten and can't eat until all their adjacent philosophers (and ultimately P_i) have eaten as well.

The proof is similar to the one of Theorem 3. The invariant in this case is

$$T_i \cap C_{i,r} \cap W_{i,s} \xrightarrow[p]{F} \begin{array}{l} (T_i \cap C_{i,r+1} \cap W_{i,s}) \\ \cup \\ (T_i \cap C_{i,r} \cap W_{i,s+1}) \cup E_i \end{array}$$

where p has the same lower bound as in the proof of Theorem 3. Furthermore, if h is the total number of cycles containing P_i, and m is the total number of philosophers connected to P_i, we have

$$T_i \cap C_{i,h} \cap W_{i,s} \xrightarrow[p]{F} (T_i \cap C_{i,h} \cap W_{i,s+1}) \cup E_i,$$

$$T_i \cap C_{i,r} \cap W_{i,m} \xrightarrow[p]{F} (T_i \cap C_{i,r+1} \cap W_{i,m}) \cup E_i,$$

and

$$T_i \cap C_{i,h} \cap W_{i,m} \xrightarrow[1]{F} E_i.$$

Hence, by Lemma 1 and 2 we derive

$$T_i \xrightarrow[p^{h+m}]{F} E_i.$$

Since T_i *unless* E_i, by Lemma 3 we conclude. $\square$

6. CONCLUSION AND FUTURE WORK

We have shown that the randomized algorithms of Lehmann and Rabin for the symmetric and fully distributed dining philosophers problem do not work anymore when we relax the condition that the topology is a simple ring. We have then proposed randomized solutions ensuring progress and lockout-freedom for the general case of an arbitrary connection graph.

In this paper we have focused on the existence of a solution, and we have not address any efficiency issue. Clearly, efficiency is an important attribute for an algorithm. The evaluation of the complexity of our algorithms, and possibly the study of more efficient variants, are open topics for future research.

Another open problem that seems worth exploring is the symmetric and fully distributed solution in the even more general case of hypergraphs-like connection structures, in which a philosopher may need more than two forks to eat.

This work is part of a project which aims at providing a fully distributed implementation of the π-calculus. The algorithm presented here will serve for solving the conflicts associated to the competition for channels arising in presence of guarded-choice commands.

7. ACKNOWLEDGEMENTS

The authors would like to thank Dale Miller and the anonymous referees of PODC 2001 for their helpful comments.

8. REFERENCES

[1] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, Oct. 1971. Reprinted in *Operating Systems Techniques*, C.A.R. Hoare and R.H. Perrot, Eds., Academic Press, 1972, pp. 72–93. This paper introduces the classical synchronization problem of Dining Philosophers.

[2] N. Francez and M. Rodeh. A distributed abstract data type implemented by a probabilistic communication scheme. In *Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science*, pages 373–379, 1980.

[3] O. M. Herescu and C. Palamidessi. Probabilistic asynchronous π-calculus. In J. Tiuryn, editor, *Proceedings of FOSSACS 2000 (Part of ETAPS 2000)*, Lecture Notes in Computer Science, pages 146–160. Springer-Verlag, 2000.

[4] N. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Symposium on Principles of Distributed Computing (PODC '94)*, pages 314–323, New York, USA, Aug. 1994. ACM Press.

[5] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 & 41–77, 1992.

[6] U. Nestmann. What is a 'good' encoding of guarded choice? In C. Palamidessi and J. Parrow, editors, *Proceedings of EXPRESS '97: Expressiveness in Concurrency (Santa Margherita Ligure, Italy, September 8–12, 1997)*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1997. Full version to appear in *Information and Computation*.

[7] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 256–265, Paris, France, 1997.

[8] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. The MIT Press, 1998.

[9] M. O. Rabin and D. Lehmann. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, chapter 20, pages 333–352. Prentice Hall, 1994. An extended abstract appeared in the *Proceedings of POPL'81*, pages 133-138.

[10] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

[11] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995. An extended abstract appeared in *Proceedings of CONCUR '94*, LNCS 836: 22–25.