An Efficient Starvation-Free Semaphore Solution for the Graphical Mutual Exclusion Problem

K. YUE[1]* AND R. T. JACOB[2]

[1] *Department of Computer Science, University of Houston – Clear Lake, 2700 Bay Area Boulevard, Houston, TX 77058–1098, USA*

[2] *Department of Computer Science, University of North Texas, Denton, TX 76203–3886, USA*

A fast method for constructing efficient solutions for graphical mutual exclusion problems based on semaphores associated with processes is described. The number of semaphores used is equal to the number of processes in the mutual exclusion problem. The solution is both deadlock-free and starvation-free, and allows a reasonable degree of concurrency. This method can be generalised to deal with generalised semaphore systems such as the PV_{chunk}.

Received April 1989, revised July 1989

1. INTRODUCTION

In a graphical mutual exclusion problem,[9] there are a finite number of processes, all assumed, without loss of generality, to have the following code:

```
loop forever
    entry section
    critical section
    exit section
    non-critical section
end loop
```

The critical section of a process is mutually exclusive to the critical sections of one or more other processes, but not necessarily all.[11] There are well-known solutions to some special problems such as the Readers and Writers Problem or the Dining Philosophers Problem.[1,2,3] However, with few exceptions, the semaphore solutions of mutual exclusion problems with general topology are not well studied.

In Ref. 7, a method for generating efficient deadlock-free solutions for a wide class of problems, known as *edge-solvable problems*, was presented. This method was extended for the generation of starvation-free solutions in Refs. 9, 12 and 13. However, it cannot be applied to problems that are not edge-solvable, such as the Dining Philosophers Problem.

For general problems, a straight-forward method is to simulate the action of a FIFO Hoare monitor.[1,6,9,11] Since only one process can be inside the monitor at any time, it is easy to construct starvation-free solutions. However, these monitor-like solutions are inefficient:[9,11]

(1) Either a large number of counting variables is used ($2N+1$ in Ref. 11, where N is the number of processes in the problem), or a shared queue with complicated state handling for each process is necessary.[9]

(2) The code for the entry and exit section is long. Much time is spent on either updating the counting variables or maintaining the queue and states.

(3) A gate semaphore must be used to guard the entry to the monitor, so no two processes can progress in their entry sections at the same time. This can easily be a serious bottleneck limiting concurrency.

In Refs 12 and 13, another method was proposed which associates every mutual exclusion constraint with a unique semaphore. This *edge-associated solution* is good for any problem and does not have the drawback of the monitor-like solutions. However, the number of semaphores used is equal to N_e, the number of mutual exclusion constraints in the problem. This number is $O(N^2)$, too large to be practical for complicated problems. In this paper, we propose a *node-associated solution* which has most of the advantages of the edge-associated solution and uses only N semaphores.

2. NODE-ASSOCIATED SOLUTIONS

Following Refs 7 and 9, a graphical mutual exclusion problem is represented by a graph $G(V, E)$ where V and E are the sets of nodes and edges respectively. A process in the problem is represented by a node in V and a mutual exclusion constraint between a pair of processes is represented by an edge, joining the corresponding nodes, in E. The terms 'node' and 'process' are used interchangeably. So are the terms 'graph' and 'problem'.

Throughout this paper, strong semaphores in Ref. 10 are used. Hence, it is not possible to have individual *competition starvation* in which a process waiting for the completion of a P operation can be overtaken by other processes infinitely many times.[10] It is only necessary to consider *no-food starvation*[12,13] where a process is starved by waiting at a P operation with no corresponding V operation being issued. As an example, Figure 1 shows a well-known solution for the Readers and Writers Problem where no-food starvation can occur to a Writer when there is a continuous stream of Readers such that N_Readers is never reset to 0.

```
Writers
    entry section: P(Mutex);
    exit section:  V(Mutex);
Readers
    entry section: P(Gate);
                   N_Readers ← N_Readers + 1;
                   if N_Readers = 1 then P(Mutex);
                   V(Gate);
    exit section:  P(Gate);
                   N_Readers ← N_Readers − 1;
                   if N_Readers = 0 then V(Mutex);
                   V(Gate);
```

Figure 1. A starvation-possible solution for the Readers and Writers Problem.

* To whom correspondence should be addressed.

Following and extending Ref. 7, the following minimum requirements on the solutions are made.

(1) (*Mutual Exclusion Constraint*) Two neighbouring nodes cannot be in their critical sections at the same time.

(2) (*Absence of Deadlock Constraint*) No deadlock is allowed.

(3) (*Absence of Starvation Constraint*) No starvation of any process is allowed.

(4) (*Concurrency Constraint*) It is possible for two non-neighbouring nodes to be in their critical sections at the same time.

The concurrency constraint ensures that the solution should allow a reasonable degree of concurrency. This is important since a scheme allowing at most one process in its critical section can always satisfy the first three constraints but is extremely inefficient.

The concept of node-associated solutions is developed below. The set of neighbours for a node p is denoted by $N(p)$. A semaphore X in a solution is said to be a *node-associated semaphore* of a node p (written as S_p) if it is used in the synchronisation sections of p and its neighbour, but of no other node.

A solution of a problem $G(V, E)$ is said to be a *node-associated solution* if:

(1) there is a node-associated semaphore for every node in V;

(2) the entry section of any node p consists solely of (*a*) a sequence of P operations on the node-associated semaphores of p and all nodes in $N(p)$, followed by (*b*) a sequence of V operations on the node-associated semaphores of all nodes in $N(p)$, and;

(3) The exit section of any node p consists solely of the operation $V(S_p)$.

As an example, consider the graph in Figure 2:

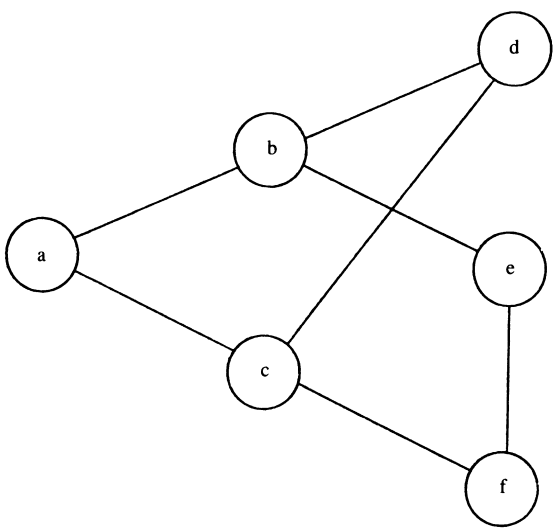

Figure 2. A mutual exclusion graph *G*.

$G(\{a, b, c, d, e, f\},$
$\{(a, b), (a, c), (b, d), (b, e), (c, d), (c, f), (e, f)\}).$

Figure 3 shows a node-associated solution for G. All three conditions for a node-associated solution are satisfied. In contrast, Figure 4 shows an edge-associated solution for G as generated by Ref. 12.

The following lemmas summarise the property of a node-associated solution.

Node a
entry section: $P(S_a)$; $P(S_b)$; $P(S_c)$; $V(S_c)$; $V(S_b)$;
exit section: $V(S_a)$;
Node b
entry section: $P(S_a)$; $P(S_b)$; $P(S_d)$; $P(S_e)$; $V(S_e)$; $V(S_d)$; $V(S_a)$;
exit section: $V(S_b)$;
Node c
entry section: $P(S_a)$; $P(S_c)$; $P(S_d)$; $P(S_f)$; $V(S_f)$; $V(S_d)$; $V(S_a)$;
exit section: $V(S_c)$;
Node d
entry section: $P(S_b)$; $P(S_c)$; $P(S_d)$; $V(S_c)$; $V(S_b)$;
exit section: $V(S_d)$;
Node e
entry section: $P(S_b)$; $P(S_e)$; $P(S_f)$; $V(S_f)$; $V(S_b)$;
exit section: $V(S_e)$;
Node f
entry section: $P(S_c)$; $P(S_e)$; $P(S_f)$; $V(S_e)$; $V(S_c)$;
exit section: $V(S_f)$;

Figure 3. A node-associated solution for the graph *G*.

Node a
entry section: $P(S_{ab})$; $P(S_{ac})$;
exit section: $V(S_{ac})$; $V(S_{ab})$;
Node b
entry section: $P(S_{ab})$; $P(S_{bd})$; $P(S_{be})$;
exit section: $V(S_{be})$; $V(S_{bd})$; $V(S_{ab})$;
Node c
entry section: $P(S_{ac})$; $P(S_{cd})$; $P(S_{cf})$;
exit section: $V(S_{cf})$; $V(S_{cd})$; $V(S_{ac})$;
Node d
entry section: $P(S_{bd})$; $P(S_{cd})$;
exit section: $V(S_{cd})$; $V(S_{bd})$;
Node e
entry section: $P(S_{be})$; $P(S_{ef})$;
exit section: $V(S_{ef})$; $V(S_{be})$;
Node f
entry section: $P(S_{cf})$; $P(S_{ef})$;
exit section: $V(S_{ef})$; $V(S_{cf})$;

Figure 4. An edge-associated solution for the graph *G*.

Lemma 1. If a node-associated solution is deadlock-free, then it is also starvation-free.

Proof. It is only necessary to consider no-food starvation. We will prove that if starvation exists, then deadlock always exist.

Suppose a node is starved at $P(S_x)$. Note that in a node-associated solution, if a node q proceeds after having completed a $P(S_x)$, it will eventually issue a $V(S_x)$ either in its entry section or in its exit section. The only ways that q could not eventually issue a $V(S_x)$ are: (1) q is involved in a deadlock, or (2) q is itself starved. In (1), we have a deadlock and the lemma is proved. In (2), q must be starved at some $P(S_y)$, completed earlier by some node r. Similarly, r can either be involved in a deadlock or be starved at some $P(S_z)$, completed earlier by some node s. Continuing in this manner, since the

number of nodes is finite, either (1) all nodes in a proper subset of V are involved in a deadlock or (2) all nodes in V are starved and waiting at P operations. However, case (2) is equivalent to a deadlock involving all nodes. □

Lemma 2. A node-associated solution satisfies the mutual exclusion constraint and the concurrency constraint.

Proof. The entry sections of two neighbouring nodes p and q both contain $P(S_p)$ and $P(S_q)$. Consider a time t when p has completed all its P operations but has not started signalling the V operations in its entry section. At this time, node q cannot be in its critical section, since, otherwise, p would not have been able to complete $P(S_q)$. Furthermore, if q is in its entry section, then q must not have completed $P(S_p)$ since p has already completed it. Therefore, if q wants to enter its critical section at time greater than t, it must complete $P(S_p)$ in its entry section. Since p signals $V(S_p)$ only in its exit section, q cannot complete $P(S_q)$ so long as p is in its critical section. Hence, two neighbouring nodes p and q cannot be at their critical sections at the same time.

For the concurrency constraint, consider the situation where only node p is in its critical section and all other nodes are in their non-critical sections. In this case, only the semaphore S_p is set to 0. Hence, if a non-neighbouring node r wants to enter its critical section while all other process remains in their same section, r is allowed to do so since $P(S_p)$ does not appear in the entry section of r. Therefore, it is possible for two non-neighbouring nodes to be in their critical sections at the same time. □

Although it may appear natural to put all V operations in the exit sections, it is important to put all V operations on the semaphores associated with the neighbours of a node p in the entry section of p. Otherwise, the concurrency constraint may not be satisfied. As an example, Figure 5 shows a solution obtained from Figure 3 by moving all V operations to the exit sections. In this case, if node a is in its critical section, S_b must be decremented to 0. Therefore, although d is not a neighbour of node a, it is not possible for d to enter its critical section since $P(S_b)$ is included in the entry section of d.

Node a
entry section: $P(S_a)$; $P(S_b)$; $P(S_c)$;
exit section: $V(S_c)$; $V(S_b)$; $V(S_a)$;
Node b
entry section: $P(S_a)$; $P(S_b)$; $P(S_d)$; $P(S_e)$;
exit section: $V(S_e)$; $V(S_d)$; $V(S_b)$; $V(S_a)$;
Node c
entry section: $P(S_a)$; $P(S_c)$; $P(S_d)$; $P(S_f)$;
exit section: $V(S_f)$; $V(S_d)$; $V(S_c)$; $V(S_a)$;
Node d
entry section: $P(S_b)$; $P(S_c)$; $P(S_d)$;
exit section: $V(S_d)$; $V(S_c)$; $V(S_b)$;
Node e
entry section: $P(S_b)$; $P(S_e)$; $P(S_f)$;
exit section: $V(S_f)$; $V(S_e)$; $V(S_b)$;
Node f
entry section: $P(S_c)$; $P(S_e)$; $P(S_f)$;
exit section: $V(S_f)$; $V(S_e)$; $V(S_c)$;

Figure 5. A solution for the graph G that does not satisfy the Concurrency Constraint.

3. CODE GENERATION

The following algorithm generates a node-associated solution, in Pascal-like syntax, for any general mutual exclusion problem. The string concatenation operator and the set union operator are denoted by & and U respectively.

Algorithm 1. Generation of starvation-free node-associated semaphore solutions for mutual exclusion problems.

Input. A graph $G(V, E)$ with N nodes.

Output. The entry and exit sections, stored in entry(p) and exit(p) respectively, of all nodes p in V.

Semaphores used. Every node p has a semaphore S_p, initially 1, associated with it.

(1) Arbitrarily label each node in V with an unique integer value from 1 to N.

(2) **For** every node p in V **do**
work_list ← $N(p) \cup \{p\}$;
V_codes ← " ";
entry(p) ← " ";
exit(p) ← " $V(S_p)$;";
while *work_list* is not *empty* **do**.
Let x be the node in work_list labelled with the smallest value;
entry(p) ← *entry*(p) &" $P(S_x)$;";
if $x <> p$ **then** *V_code* ← *V_code* & " $V(S_x)$;";
Remove x from work_list;
end *while*;
entry(p) ← *entry*(p) & *V_code*(p);
end *for*.

(3) Stop.

As an example of Algorithm 1, the solution of Figure 3 is generated for the graph G of Figure 2 if the values of 1, 2, 3, 4, 5 and 6 are labelled to nodes a, b, c, d, e and f respectively.

Lemma 3. The solution generated by Algorithm 1 is deadlock-free.

Proof. Suppose it is possible to construct a deadlock situation involving all nodes in some D, a non-empty subset of V. Let X be the semaphore with the highest value, labelled by Algorithm 1, among all semaphores blocking the nodes in D. There must be some node in D, say q, that has completed $P(X)$. Suppose q is itself blocked at $P(Y)$. In that case, $P(Y)$ must appear later than $P(X)$ in the entry section of q. Because of the way of code being generated for the entry section of q, Y must be labelled with a value greater than X, contradicting to the assumption that X has the highest labelled value among all blocking semaphores in D. □

Theorem 1. Algorithm 1 generates starvation-free solutions satisfying all constraints 1 to 4.

Proof. By combining Lemmas 1 to 3. □

Compared with monitor-like solutions, the node-associated solutions have several major advantages.

(1) There are no counting variables or shared queue and no related update overhead.

(2) Entry and exit sections are short.

(3) Several processes can progress in their entry sections.

There is no universal gate semaphore and hence no corresponding bottleneck.

Furthermore, the node-associated solution uses only N semaphores, as opposed to N_e semaphores in the edge-associated solution. Since the value of N_e may range from $N-1$ to $N*(N-1)/2$ in a connected graph, the difference in the number of semaphores can be quite significant, especially important when the semaphores are implemented in scarce shared memory.

However, it should be noted that the edge-associated solutions allow more concurrency than the node-associated solutions. This is because a semaphore is always used by only two processes in edge-associated solutions but can be used by more than two processes in the node-associated solutions.

As an example, consider the graph G of Figure 2 and its two solutions in Figures 3 and 4. Suppose node f is in its critical section and all other nodes are in non-critical sections. If node e wants to enter its critical section, it will be blocked either at $P(S_{ef})$ in the edge-associated solution of Figure 4 or at $P(S_f)$ in the node-associated solution of Figure 3. If node a now wants to enter its critical section, it will be allowed to do so in the edge-associated solution but will be blocked at $P(S_b)$ in the node-associated solution since node e had already completed $P(S_b)$. This example shows that, in a node-associated solution, if a neighbour of a neighbour of a node p is already blocked, then p may also be blocked if it tries to execute its entry section.

4. GENERALISED PV SYSTEMS

Algorithm 1 can easily be extended for many generalised PV systems.[4,5,8] In a PV_{chunk} (or simply PV_c) system, the value of a semaphore can be incremented or decremented by any positive integer, not necessarily 1. The operations P_c and V_c are defined as:

$P_c(S:a)$: **if** $S >= a$ **then** $S \leftarrow S-a$
else *wait*.
$V_c(S:b)$: $S \leftarrow S+b$.

Using the PV_c system, it is possible to modify Algorithm 1 so that all V operations appear in the exit sections.

Algorithm 2. Generation of starvation-free node-associated PV_c semaphore solutions for mutual exclusion problems.

Input. A graph $G(V, E)$ with N nodes.

Output. The entry and exit sections, stored in entry(p) and exit(p) respectively, of all nodes p in V.

Semaphores used. Every node p in V has a semaphore S_p associated with it. S_p is initialised to N_p, the number of neighbours of p.

(1) Arbitrarily label each node in V with an unique integer value from 1 to N.

(2) **For** every node p in V **do**
work_list $\leftarrow N(p) \cup \{p\}$
entry(p) $\leftarrow$ " ";
exit(p) $\leftarrow$ " ";

while *work_list* is not *empty* **do**
Let x be the node in work_list labelled with the smallest value;
if $x = p$ **then**
entry(p) $\leftarrow$ *entry*(p) & " $P(S_p:N_p)$; ";
exit(p) $\leftarrow$ " $V(S_p:N_p)$;" & *exit*(p);
else
entry(p) $\leftarrow$ *entry*(p) & " $P(S_x:1)$; ";
exit(p) $\leftarrow$ " $(S_x:1)$;" & *exit*(p);
end *if*;
Remove x from work_list;
end ***while;***
end ***for***.

(3) Stop.

Using Algorithm 2, the solution in Figure 6 is generated for the graph G of Figure 2.

Node a
entry section: $P(S_a:2)$; $P(S_b:1)$; $P(S_c:1)$;
exit section: $V(S_c:1)$; $V(S_b:1)$; $V(S_a:2)$;
Node b
entry section: $P(S_a:1)$; $P(S_b:3)$; $P(S_d:1)$; $P(S_e:1)$;
exit section: $V(S_e:1)$ $V(S_d:1)$; $V(S_b:3)$; $V(S_a:1)$;
Node c
entry section: $P(S_a:1)$; $P(S_c:3)$; $P(S_d:1)$; $P(S_f:1)$;
exit section: $V(S_f:1)$; $V(S_d:1)$; $V(S_c:3)$; $V(S_a:1)$;
Node d
entry section: $P(S_b:1)$; $P(S_c:1)$; $P(S_d:2)$;
exit section: $V(S_d:2)$; $V(S_c:1)$; $V(S_b:1)$;
Node e
entry section: $P(S_b:1)$; $P(S_e:2)$; $P(S_f:1)$;
exit section: $V(S_f:1)$; $V(S_e:2)$; $V(S_b:1)$;
Node f
entry section: $P(S_c:1)$; $P(S_e:1)$; $P(S_f:2)$;
exit section: $V(S_f:2)$; $V(S_e:1)$; $V(S_c:1)$;

Figure 6. A PV_c node-associated solution for the graph G.

The correctness of the solutions generated by Algorithm 2 can be proved in a manner similar to that of Algorithm 1 and is omitted here. These solutions allow more concurrency than that of Algorithm 1. For example, a node may still enter its critical section even if a neighbour of one of its neighbour is blocked earlier. The degree of concurrency allowed is similar to that of the edge-associated solutions.

In a similar fashion, Algorithm 1 can also be extended to deal with PV_{multiple} system or the PV_{general} system.[5] The details are not pursued here.

5. CONCLUSION

In this paper, an algorithm is presented that generates starvation-free semaphore solutions for graphical mutual exclusion problems. These solutions are much more efficient than the straightforward monitor-like solutions. Although they are slightly more restrictive than the edge-associated solutions, they usually use a significantly smaller number of semaphores, making them especially suitable for complicated problems.

However, while the edge-associated solutions are also applicable for *weak semaphores*,[10,13] where individual competition starvation is possible, the node-associated solutions are not. It would be interesting to obtain an extension of Algorithm 1 that accommodates weak semaphores.

Acknowledgement

The authors would like to express their thanks to the Faculty Research and Support Funds of the University of Houston – Clear Lake and to the referee for his various valuable suggestions for improvement.