A priority dynamics for generalized drinking philosophers

Valmir C. Barbosa a,*, Mario R.F. Benevides b, Ayru L. Oliveira Filho c

a Universidade Federal do Rio de Janeiro, COPPE, Caixa Postal 68511, 21945-970 Rio de Janeiro, RJ, Brazil
b Universidade Federal do Rio de Janeiro, COPPE and Instituto de Matemática, Caixa Postal 68511, 21945-970 Rio de Janeiro, RJ, Brazil
c Centro de Pesquisas de Energia Elétrica—CEPEL, Caixa Postal 68007, 21944-970 Rio de Janeiro, RJ, Brazil

Received 21 June 2000; received in revised form 28 September 2000
Communicated by M. Yamashita

Abstract

We consider a generalization of the drinking philosophers problem, called GDrPP, in which processes may issue AND-OR requests for resources, as opposed to the AND requests of the original formulation. For GDrPP, we introduce a basic priority dynamics that can be regarded as generalizing the edge-reversal priority dynamics underlying the classical solution to the original formulation. The novel priority dynamics is based on new results on the graphs that reflect waits due to AND-OR requests.

Keywords: Drinking philosophers problem; AND-OR resource requests; Concurrency; Distributed computing; Operating systems

1. Introduction

We consider a set P of processes and a set R of resources that can only be used by processes under the condition of mutual exclusion. For \( p_i \in P \), \( R_i \subseteq R \) comprises all the resources that process \( p_i \) may ever use in any resource-sharing computation in which it participates. These sets indicate the maximum resource usage of the processes, and can be used to construct an undirected graph, call it \( G \), that represents the sharing of resources by the processes. Graph \( G \) has \( P \) for node set and has an undirected edge between nodes \( p_i \) and \( p_j \) if and only if there exists the possibility that \( p_i \) and \( p_j \) ever become interested in the same resource concurrently, that is, \( R_i \cap R_j \neq \emptyset \).

The dining philosophers problem (DPP) [6,7] is a resource-sharing problem on \( G \) that asks for a policy to be devised for processes to share resources in such a way that mutual exclusion, deadlock-freedom, and lockout-freedom are ensured at all times. In DPP, a process \( p_i \) always requests access to all the resources in \( R_i \) concomitantly. This problem is generalized by the drinking philosophers problem (DrPP) [6], in which resource-sharing policies must be devised satisfying the same constraints as in DPP, but \( p_i \) may request access to any nonempty subset of \( R_i \) whenever in need for shared resources.

DrPP can be solved in a variety of ways [1,6,12], but the solutions that are of interest to this paper work by assigning relative priorities to the processes and then using those priorities to allocate resources to them when conflicts arise. One particularly interesting priority scheme is the one introduced in [6], which

* Corresponding author. Current address: University of California, Computer Science Division, 329 Soda Hall, Berkeley, CA 94720, USA.
E-mail addresses: valmir@cos.ufrj.br, valmir@cs.berkeley.edu (V.C. Barbosa), mario@cos.ufrj.br (M.R.F. Benevides), ayru@cepel.br (A.L. Oliveira Filho).
works by creating a directed version of $G$. The resulting directed graph, denoted by $\tilde{G}$, is required to be acyclic (have no directed cycles) and represents priorities as follows. If processes $p_i$ and $p_j$ are neighbors in $G$ and in $\tilde{G}$ the edge between them is directed from $p_i$ to $p_j$, then $p_j$ has priority over $p_i$ in the use of all the resources that they share.

If processes always use resources for a finite period of time, then the following is an outline of a solution to DrPP that employs the priority scheme based on $G$ [6]. When $p_i$ needs to use some of the resources in $R_i$, it sends requests for the resources it lacks to those of its neighbors with which they are shared. Process $p_j$, upon receiving such a request, either grants access to the resource immediately (if it is not using the resource, nor does it need the resource and has priority over $p_i$), or within a finite time (if it is using the resource), or else postpones granting access to when it successfully acquires and uses the resources it needs. The latter possibility happens when $p_j$ also needs the resource requested by $p_i$ and in addition has priority over $p_i$ to use that resource.

Because $\tilde{G}$ is acyclic, at least one process is guaranteed to be successful in acquiring and using the resources it needs (no deadlocks occur). But $\tilde{G}$ represents a static priority assignment, so it is conceivable that lockouts happen. In order to overcome this, the following priority dynamics, known as scheduling by edge reversal (SER), is used. After process $p_i$ has succeeded in acquiring and using the resources it needs, a local change in $G$ is effected to yield $\tilde{G}'$, a new directed graph. This local change is the reversal of the directions of all edges incident to $p_i$ that are directed toward $p_i$. Graph $\tilde{G}'$ is guaranteed to be acyclic [6], and therefore constitutes a new priority scheme, one in which the priorities between $p_i$ and some of its neighbors have been reversed. Lockouts are then guaranteed never to occur because processes eventually move up in the priority hierarchy.

The net effect of SER is to provide a priority dynamics that underlies all the wait for resources in $G$. In any consistent global state (in the sense of [1, 5]) of the resource-sharing computation, process $p_i$ is waiting for process $p_j$ if and only if $p_i$ has sent $p_j$ a request but $p_j$ has not granted that request because either it is using the resource under consideration or it too needs the resource and furthermore has priority over $p_i$ for use of that resource. If we disregard waits of the former type (because they necessarily end after a finite period of time), then the wait-for graph, call it $W$, that represents processes’ waits in this global state is a directed graph whose edges coincide with some of the edges of $\tilde{G}$, the graph that gives priorities in that same global state. So, once a global state is fixed, $W$ is a subgraph of $\tilde{G}$. Because $\tilde{G}$ is always acyclic, then so is $W$, which is the well-known necessary and sufficient condition for deadlocks never to occur. Not only this, but in the evolving $W$ longest directed paths from fixed nodes become ever shorter, which is a measure of how close processes are to acquiring the resources they need, hence lockouts never occur, either.

But this holds only because in DrPP requests for resources are always of the so-called AND type of requests. That is, what processes request are conjunctions of resources. For AND requests, SER provides a means to prevent deadlocks by ensuring that wait-for graphs are always acyclic; it prevents the occurrence of lockouts by ensuring that, eventually, a process has priority over all its competitors.

In this paper, we introduce a generalization of DrPP, called the generalized drinking philosophers problem (GDrPP), in which requests for resources are AND-OR requests. With requests of this type, processes request disjunctions of conjunctions of resources (either one group of resources, or another group, or yet another, and so on). Resource-sharing computations with AND-OR requests have been studied extensively, especially from the perspective of deadlock detection [4, 9, 11]. The central question that we address in this paper is the question of generating a priority dynamics that can do for AND-OR requests what SER does for AND requests, namely, prevent the occurrence of deadlocks while ensuring the absence of lockouts as well.

With AND-OR requests, process $p_i$ sends out requests to groups of neighbors in $G$ representing the conjunctions of resources it needs. Only from processes in one of these groups does $p_i$ need to receive grant messages, because its demand for resources is satisfied by a disjunction of those conjunctions. The interaction between $p_i$ and its neighbors takes place in much the same way as described for AND requests, although now $p_i$ also sends relinquish messages to processes in groups other than the one from whose members it receives the grant messages that satisfy its demand.
The following is how the remainder of the paper is organized. Section 2 contains an analysis of the wait-for graph as it relates to a priority structure under AND-OR requests. This analysis provides the formal basis for the priority dynamics that we introduce in Section 3. Like SER, this dynamics too works on the directed version \( \tilde{G} \) of \( G \). It is called scheduling by selective edge reversal (SSER), and provides deadlock- and lockout-free rearrangements of priorities for GDrPP. Concluding remarks are given in Section 4.

2. The wait-for graph

Similarly to the case of AND requests, in any consistent global state of a resource-sharing computation with AND-OR requests, process \( p_i \) is waiting for process \( p_j \) if and only if \( p_i \) has sent \( p_j \) a request, has not received from \( p_j \) a grant response (for the same reasons as before, of which we also disregard waits due to current use), and furthermore has not sent \( p_j \) a message relinquishing interest in the resources it requested. As in the case of AND requests, the wait-for graph \( W \) is a subgraph of the priority graph \( \tilde{G} \) in that global state. Unlike that case, however, the existence of directed cycles in \( W \), although still necessary for the occurrence of deadlocks, is no longer sufficient. In this section, we establish the formal basis for the remainder of the paper by demonstrating properties of \( W \) that can be used to characterize deadlocks under AND-OR requests [2,10]. In particular, we give a tighter necessary condition for deadlocks to occur than simply the existence of directed cycles in \( W \).

If \( p_i \) is a node of \( W \), then let \( O_i \) denote the set of nodes in \( W \) toward which an edge is directed from \( p_i \). If \( O_i \) is nonempty, then, for \( t_i \geq 0 \), its nodes can be partitioned into nonempty sets \( W_1^i, \ldots, W_{\ell_i}^i \) characterizing the wait of \( p_i \); the process is either waiting for all processes in \( W_1^i \), or all processes in \( W_2^i \), and so on. We assume that none of these sets is a subset of another, that is, the AND-OR requests issued by \( p_i \) are not redundant. Note that this assumption must hold for the wait-for graph in all consistent global states. For this reason, whenever changes in the graph cause the appearance of sets \( W_k^i \) and \( W_{k'}^i \) such that \( W_k^i \subseteq W_{k'}^i \) for \( 1 \leq k, k' \leq t_i \), we assume that the wait represented by the redundant \( W_k^i \) is eliminated through the removal of all superfluous edges directed from \( p_i \) toward nodes in \( W_k^i \). If \( p_i \) is free to compute on shared resources (so \( O_i \) is necessarily empty), then let \( \tilde{W} \) be the wait-for graph resulting from the sending by \( p_i \) of grant messages to every \( p_j \) such that \( p_i \in O_j \), if any, after its computation on shared resources has ended. We say that \( \tilde{W} \) is message-reduced from \( W \) by \( p_i \). After the process of message-reduction, \( p_i \) is an isolated node and the aforementioned eliminations may happen for \( p_j \) such that \( p_i \in O_j \).

Consider, for example, the illustration given in Fig. 1, in which part (a) depicts \( \tilde{W} \) and part (b) depicts \( \bar{W} \), the latter being message-reduced from \( \tilde{W} \) by \( p_1 \), (circular arcs are used to indicate the AND waits of nodes). Note that not only does \( p_1 \) become isolated in \( \bar{W} \), but also the edge directed from \( p_6 \) to \( p_5 \) need no longer exist (once a grant message is received by \( p_6 \) from \( p_1 \), the only further message that it needs is from \( p_2 \), as a message from \( p_5 \) would be irrelevant to its wait condition).

We consider two types of subgraphs of \( \tilde{W} \), called b-subgraphs and c-subgraphs. A subgraph \( \tilde{H} \) of \( \tilde{W} \) is a b-subgraph if and only if, for every node \( p_i \) of \( \tilde{H} \) for which \( O_i \) is nonempty, the edges that in \( \tilde{H} \) are directed away from \( p_i \) are such that at least one leads to each of \( W_1^i, \ldots, W_{\ell_i}^i \). It is called a c-subgraph if and only if the edges that in \( \tilde{H} \) are directed away from \( p_i \) lead to all nodes of exactly one of \( W_1^i, \ldots, W_{\ell_i}^i \). Intuitively, the global wait that \( \tilde{W} \) represents is the conjunction of all waits represented by its b-subgraphs (all OR waits must be relieved), or the disjunction of all waits represented by those of its c-subgraphs having the same node set as itself (at least one graph-wide AND wait must be relieved).

Figs. 2–5 illustrate these subgraphs of \( \tilde{W} \). Parts (b) and (c) of Fig. 2 contain b-subgraphs of \( \tilde{W} \) of part (a). Two of its c-subgraphs are shown in parts (a)
Fig. 2. $\vec{W}$ with two b-subgraphs.

Fig. 3. Two c-subgraphs of the $\vec{W}$ of Fig. 2(a).

and (b) of Fig. 3. Similarly for Figs. 4 and 5 with respect to the $W$ of Fig. 4(a).

A few more definitions are in order. A sink in a directed graph is any node whose incident edges are all directed inward. Clearly, in the case of $\vec{W}$, $p_i$ is a sink if and only if $O_i = \emptyset$, so only sinks are free to compute on shared resources and eventually send grant messages. A knot is a set $K$ of nodes with the properties that $|K| > 1$ and that it is the set of nodes reachable in the directed graph from each of the nodes in $K$. When only OR requests are employed, the presence of a knot in $\vec{W}$ is necessary and sufficient for a deadlock to exist [8]. This can be generalized for AND-OR requests as in Theorem 2, given after the following supporting result.

Lemma 1. If no b-subgraph of $\vec{W}$ has a knot, then let $p_i$ be a sink in $\vec{W}$ and let $W'$ be message-reduced from $W$ by $p_i$. Then no b-subgraph of $\vec{W}'$ has a knot.

Proof. Let $\vec{H}'$ be a b-subgraph of $\vec{W}'$. We show that $\vec{H}'$ (either by itself or enlarged by a single edge directed toward $p_i$) must also be a b-subgraph of $\vec{W}$, having therefore no knots, by hypothesis. There are four cases to be considered. First, if no $p_j$ such that $p_i \in O_j$ is a node of $\vec{H}'$, then clearly $\vec{H}'$ is also a b-subgraph of $\vec{W}$. Otherwise, let $p_j$ be a node in $\vec{H}'$ such that $p_i \in O_j$, and assume that $|O_j| > 1$. In the second case, the waits represented by all the sets $W_j^1, \ldots, W_j^{t_j}$ remain valid after the message-reduction from $p_i$, so in $\vec{H}'$ there exists at least one edge directed away from $p_j$ toward each of those sets, and again $\vec{H}'$ is a b-subgraph of $\vec{W}$. The third case corresponds to the existence of $W_{j,k}^k$ and $W_{j,k}'$ with $1 \leq k, k' \leq t_j$ such that $\{p_i\} = W_{j,k}' \setminus W_{j,k}^k$. Although the wait represented by $W_{j,k}^k$ is eliminated by the message-reduction, $\vec{H}'$ has an edge directed from $p_j$ to some node in $W_{j,k}' \setminus \{p_i\}$, which is a subset of $W_{j,k}^k$, and therefore $\vec{H}'$ is also in this case a b-subgraph of $\vec{W}$. The fourth and final case corresponds to $|O_j| = 1$. In this case, $\vec{H}'$ has no edges directed outward from $p_j$, so it is not a b-subgraph of $\vec{W}$. However, enlarging $\vec{H}'$ by a single edge directed from $p_j$ toward $p_i$ does yield a b-subgraph of $W$, thus completing the proof.

Theorem 2. There exists a deadlock in $\vec{W}$ if and only if at least one of the b-subgraphs of $\vec{W}$ has a knot.

Proof. If at least one of the b-subgraphs of $\vec{W}$ has a knot, then let $\vec{H}$ be such a b-subgraph. A node in this knot is blocked for the reception of a grant message from at least one of the nodes to which it connects forward by an edge in $\vec{H}$. But the existence of the knot means that its wait will never end, which characterizes a deadlock.

Conversely, suppose that none of the b-subgraphs of $\vec{W}$ has a knot. In order to prove that in this case no deadlock exists, we must show that, if $\vec{W}$ can only evolve by the removal of edges as messages are sent to unblock waiting nodes, then eventually all waits are eliminated and $\vec{W}$ stabilizes as a graph with no edges.
Fig. 4. $\vec{W}$ with two b-subgraphs.

Fig. 5. Two c-subgraphs of the $\vec{W}$ of Fig. 4(a).

But this is guaranteed directly by Lemma 1, thence the theorem. □

Next are two additional results on the presence of knots in the b-subgraphs of $\vec{W}$. They relate such knots to directed cycles in certain c-subgraphs of $\vec{W}$. But first a little additional nomenclature must be recalled.

A strongly connected component of a directed graph is a subgraph that is maximal with respect to the property that all of its nodes are reachable from one another (so every subgraph having a knot for node set is strongly connected, but not conversely). A subgraph is said to be spanning if and only if its node set is the same as that of the original graph.

**Lemma 3.** If no b-subgraph of $\vec{W}$ has a knot, then every strongly connected component of $\vec{W}$ has at least one node $p_i$ such that at least one of $W_1^i, \ldots, W_t^i$ does not intersect the component’s node set.

**Proof.** The lemma is trivial for single-node components. If this is not the case, then let $\bar{C}$ be a strongly connected component of $\vec{W}$ with more than one node, and let $L$ be its node set. Suppose that every $p_i \in L$ is such that all of $W_1^i, \ldots, W_t^i$ intersect $L$. We show that $\vec{W}$ contains a b-subgraph $\vec{H}$ that has a knot. This b-subgraph has node set $L$, and in it each $p_i \in L$ has at least one node in each of $W_1^i \cap L, \ldots, W_t^i \cap L$. Note that, by assumption, this construction is always possible. Also, because $\bar{C}$ is strongly connected, $\vec{H}$ has no sinks. Now consider the sequence of sets $D_1^i, D_2^i, D_3^i, \ldots$, for some $p_i \in L$, such that $D_1^i$ is the set of nodes to which $p_i$ connects forward by an edge in $\vec{H}$ and, for $k > 1$, $D_k^i$ is the set of nodes in $\vec{H}$ to which the nodes in $D_1^{k-1}$ are directly connected by forward edges. The absence of sinks in $\vec{H}$ ensures that all sets in this sequence are nonempty. In addition, because $L$ is finite, the sequence has a fixed point, which is by definition a knot in $\vec{H}$. □

**Theorem 4.** At least one of the b-subgraphs of $\vec{W}$ has a knot if and only if every spanning c-subgraph of $\vec{W}$ has a directed cycle.

**Proof.** Let $K$ be a knot in some b-subgraph of $\vec{W}$. By definition, every spanning c-subgraph of $\vec{W}$ includes $K$ as part of its node set. Let $\vec{H}$ be one such c-subgraph, and consider a traversal of $\vec{H}$ that starts anywhere in $K$ and proceeds as follows. When at node $p_i$, the traversal moves on to another node to which $p_i$ connects forward by an edge in both $\vec{H}$ and the b-subgraph of $\vec{W}$ where $K$ is a knot (note that such a node must always exist, as a consequence of the very definitions of b-subgraphs and c-subgraphs). This traversal is confined to $K$ and, because $K$ is finite, must eventually return to a node already encountered, thereby characterizing a directed cycle in $\vec{H}$.

If no b-subgraph of $\vec{W}$ has a knot, then we display an acyclic spanning c-subgraph of $\vec{W}$. In order to construct such a c-subgraph, we first split the nodes of $\vec{W}$ into strongly connected components $\bar{C}_1, \ldots, \bar{C}_m$. If all of $\bar{C}_1, \ldots, \bar{C}_m$ have singletons for node sets, then
\( \bar{\mathcal{W}} \) is acyclic by the maximality of the components, and so is every one of its c-subgraphs. Otherwise, by Lemma 3, and for \( 1 \leq k \leq m \), let \( F_k \) be the nonempty set of nodes of \( \bar{\mathcal{C}}_k \) such that, if \( p_i \in F_k \), then at least one of \( W^1_i, \ldots, W^k_i \) does not intersect the node set of \( \bar{\mathcal{C}}_k \). If we regard each of \( \bar{\mathcal{C}}_1, \ldots, \bar{\mathcal{C}}_m \) as a supernode, and for all \( p_i \in F_k \) let the only edges leaving supernode \( \bar{\mathcal{C}}_k \) be those directed toward all the nodes in one of the sets \( W^1_i, \ldots, W^k_i \) that does not intersect the node set of \( \bar{\mathcal{C}}_k \), then what we have is an acyclic c-subgraph on supernodes (acyclic, as before, by the maximality of the strongly connected components). Next we shrink supernode \( \bar{\mathcal{C}}_k \) by removing \( F_k \) from its node set, and recursively repeat the entire process on what is left of \( \bar{\mathcal{C}}_k \) from the splitting into strongly connected components. The recursion ends when no such components can any longer be found that do not have singletons for node sets, at which time an acyclic spanning c-subgraph of \( \bar{\mathcal{W}} \) has been found. \( \square \)

Examples that illustrate the assertion of Theorem 4 can also be found in Figs. 2–5. Specifically, no b-subgraph of the \( \bar{\mathcal{W}} \) of Fig. 2(a) has a knot (two of them are shown in parts (b) and (c) of the figure), which is equivalent to the acyclicity of at least one of its spanning c-subgraphs (two of which are shown in Fig. 3). Similarly, the \( \bar{\mathcal{W}} \) of Fig. 4(a) has at least one b-subgraph with a knot (the one in part (b) of the figure), and this is reflected as the presence of directed cycles in all of its spanning c-subgraphs, as the one in Fig. 5(a).

3. The priority dynamics

For \( p_i \in P \) and \( m_i > 0 \), let \( M^1_i, \ldots, M^{m_i}_i \) be the sets of nodes to which \( p_i \) sends a generic AND-OR request for resources. None of these sets is a subset of another, and the requests are sent in such a way that \( p_i \)'s need will be satisfied by grant messages from all nodes in \( M^1_i \), or all nodes in \( M^2_i \), and so on. Thus, as far as the need of \( p_i \) for resources is concerned, all \( m_i \) sets are equivalent to one another. Although the value of \( m_i \) and the sets \( M^1_i, \ldots, M^{m_i}_i \) may vary from request to request, this equivalence among the \( m_i \) sets allows us to assume that there exists a fixed subset \( N_i \) of \( p_i \)'s neighbors in \( G \) that in all requests contains at least one of the sets \( M^1_i, \ldots, M^{m_i}_i \). In other words, we assume that every request issued by \( p_i \) must include a subset of \( N_i \) as one of its \( m_i \) sets. Based on this assumption, we let \( N'_i \) denote the set of neighbors \( p_j \) of \( p_i \) in \( G \) such that \( p_i \in N'_j \).

The main idea underlying SSER is to employ SER only on certain edges of the priority structure \( \bar{G} \), that is, selectively instead of indiscriminately on the edges adjacent to nodes in \( \bar{G} \) that are sinks in \( \bar{W} \). As we argue shortly, the results of Section 2 can be used to guarantee that the SER properties of deadlock- and lockout-freedom carry over to SSER as well. The following two rules summarize the operation of SSER.

1. Let \( G \) be, as in SER, a directed graph whose underlying undirected graph is \( \bar{G} \). Unlike the case of SER, \( \bar{G} \) does not have to be acyclic as a whole, but instead only its spanning subgraph in which every \( p_i \in P \) has the nodes of \( N_i \cup N'_i \) as only neighbors.

2. Let \( p_i \) be a process involved in a resource-sharing computation, \( \bar{G} \) the current priority structure on \( G \), and \( \bar{W} \) the current wait-for graph (a subgraph of \( \bar{G} \)). Let the sets that represent the wait of \( p_i \) be \( W^1_i, \ldots, W^t_i \). Each of these sets is a subset of one of \( M^1_i, \ldots, M^{m_i}_i \) (so \( t \leq m_i \)), and none of them is a subset of another. Also, by assumption, at least one of \( W^1_i, \ldots, W^t_i \) is a subset of \( N_i \). The essential SSER rule is to apply the SER rule of reversing edges outward in the following restricted manner. After \( p_i \) has succeeded in acquiring and using the resources it needs, it reverses the orientation of all edges that are in \( \bar{G} \) directed toward itself from nodes in \( N_i \cup N'_i \), thereby creating a new priority structure \( \bar{G}' \).

By Theorem 2, what has to be ensured in order for deadlock to be prevented under SSER is that no b-subgraph of a wait-for graph \( \bar{W} \) ever contains a knot. By Theorem 4, this can be done by ensuring that at least one of the spanning c-subgraphs of \( \bar{W} \) is acyclic. In SSER, this certainly holds initially, because at least the spanning c-subgraph of \( \bar{W} \) in which every \( p_i \in P \) has neighbors exclusively in \( N_i \cup N'_i \) is acyclic by rule (1). To see that it continues to hold subsequently, consider the following. Whenever \( p_i \) participates in a wait-for graph \( \bar{W} \), in at least one spanning c-subgraph of \( \bar{W} \) it has neighbors exclusively in \( N_i \cup N'_i \). If this c-subgraph is inductively assumed to be acyclic, then, by
rule (2), it is subject to the acyclicity-preserving rule of SER, and does therefore remain acyclic.

As for lockout-freedom, SSER guarantees that, in the worst case, in at least one of the spanning c-subgraphs of the evolving $\overrightarrow{W}$ a node moves progressively closer to being a sink. In that particular spanning c-subgraph, the number of edges on the longest directed path from a node to a sink is a measure of the node’s wait to become a sink, just as in SER [6], and becomes ever smaller. Because the global wait embodied by $\overrightarrow{W}$ is a disjunction of the global waits that its spanning c-subgraphs represent, that node is progressively closer to acquiring the resources it needs.

4. Concluding remarks

We have in this paper introduced SSER, which is a generalization of SER for the deadlock- and lockout-free sharing of resources under AND-OR requests. SSER operates by altering the global priority structure $\overrightarrow{G}$ in such a way that, although the acyclicity of $\overrightarrow{G}$ is not guaranteed to hold, at least one spanning c-subgraph of any wait-for graph based on $\overrightarrow{G}$ is always acyclic.

Several interesting open questions still have to be addressed, many pertaining to the choice of the $N_i$ sets for $p_i \in P$ and how it affects the performance of SSER. In particular, one such question, related to concurrency issues under SSER, is whether a concurrency analysis similar to the one carried out for SER in [3] can be undertaken.

Acknowledgements

The authors acknowledge partial support from CNPq, CAPES, the PRONEX initiative of Brazil’s MCT under contract 41.96.0857.00, a FAPERJ BBP grant, and the LOCUS Project of ProTeM/CNPq.

References