DEADLOCK AVOIDANCE IN GRAPH-STRUCTURED TASK SYSTEMS

by

M.Di MANZO, A.L.Frisiani

Istituto di Elettrotecnica
Università di Genova,Italy

and

G.Olimpo

Laboratorio per le Tecnologie Didattiche
Consiglio Nazionale Ricerche, Genova,Italy

SUMMARY

Existing models for deadlock detection and avoidance give practical solution only in the case of chains of independent tasks.
In this paper we propose a non enumerative approach to deadlock avoidance when the workload consists of a graph-structured task system. The avoidance algorithm is based on an extention of the Coffman and Denning deadlock model.

1. Introduction

The problem of deadlock avoidance has been widely examined by several authors [2,4,5,7]. However, most researches have been concerned mainly with systems of independent processes, and only a few results are known for the case of interacting processes [5] . Moreover, the deadlock detection and avoidance methods which have been defined for graph-structured task systems are essentially enumerative, and therefore quite time consuming; for practical implementation purposes sufficient and rapidly verifiable conditions are needed. On the other hand, the extensive development of parallel processing techniques and multiprocessor architecture is increasing the interest in process cooperation as a very basic tool for programming methodologies and computer systems operations; so, we feel that deadlock avoidance techniques for large systems of interacting processes should be improved.
The aim of this paper is to present a practical method for deadlock avoidance when the workload is a system of interacting processes that can be structured as a graph of tasks. With respect to the basic assumptions our model is quite close to that developed by Coffman and Denning [2]; we made this choice because the model in [2] is perhaps the most widely known and the most suitable for the extension to the case of our interest. Therefore in the following we will often refer to [2], even if a number of fundamental definitions are reported here in order to allow a self-contained reading of the paper.
The paper is organized as follows. In section 2 a set of basic definitions is presented, and a theorem is proved which allows the detection of deadlock in a general graph structured task system; in section 3 we discuss the problem of deadlock avoidance in the simplified case of a tree structured task system; in section 4 an approach to deadlock avoidance with graph structured task systems is suggested.

2. Basic definitions and theorem

We define a task system to be a pair $G = (I, \prec)$, where $I = T_1, T_2, \ldots, T_n$ is a set of tasks, and $\prec$ is a partial ordering (precedence relation) on I. Given two tasks T and T', $T \prec T'$ means that task T is to be completed before task T' begins. A task system can be represented by a precedence graph, where each vertex is a task and the vertices corresponding to two tasks T and T' are connected by a directed edge iff $T \prec T'$. A path of length k through the precedence graph G is a sequence of edges $(T_{r_1}T_{r_2})$ $(T_{r_2}T_{r_3})$ ... $(T_{r_{k-1}}T_{r_k})$ passing thorugh vertices $T_{r_1}, T_{r_2}, \ldots, T_{r_k}$.
For i and j such that $1 \leq i < j \leq k$, T_{r_i} is a predecessor of T_{r_j} and T_{r_j} is a successor of T_{r_i}; if $j=i+1$ then T_{r_i} is an immediate predecessor of T_{r_j} and T_{r_j} is an immediate successor of T_{r_i}. Let S_i be the set of immediate successors of T_i and P_i be the set of immediate predecessors of T_i. A task with no predecessors is an initial task and a task with no successors is a terminal task. The level of task T_i (written $\ell(T_i)$) is k if the length of the longest path from T_i to any terminal task is k. An execution sequence of a system G of n tasks is a sequence of task initiations and terminations $\alpha = a_1 a_2 \ldots a_{2n}$ satisfying the following conditions:

1. For every task $T \in I$ the symbols $\overline{T}$ (task initiation) and $\underline{T}$ (task termination) appear exactly once in α.
2. If $a_i = \overline{T}$ and $a_j = \underline{T}$, then $i < j$.
3. If $a_i = \underline{T}$ and $a_j = \overline{T}$, and $T \prec T'$, then $i < j$.

More details can be found in [2].
Since a task represents a computation unit during which total resource requirements don't change, the only significant events are task initiations and terminations. If the physical system consists of m resource types*, it is possible to define for each task T_i a request vector** $\hat{q}_i = (q_{i1}, q_{i2}, \ldots, q_{im})$, representing the number of units of each resource type which must be allocated to the task T_i before its initiation, a release vector $\hat{r}_i = (r_{i1}, r_{i2}, \ldots, r_{im})$, representing the number of units of each resource type which are released to the system by task T_i on its termination, and a transfer vector $\hat{t}_{ij} = (t_{ij1}, t_{ij2}, \ldots, t_{ijm})$ for each $T_j \in S_i$, representing the number of units of each resource type which are transferred to task T_j by task T_i on its termination. We suppose that every terminated task holds no unit of resource, and hence the following relation must be satisfied, for $1 \leq j \leq n$:

$$\hat{q}_j + \sum_{\forall i: T_i \in P_j} \hat{t}_{ij} = \hat{r}_j + \sum_{\forall k: T_k \in S_j} \hat{t}_{jk} \qquad (2.1)$$

---

* Obviously we will consider only those resource types which satisfy all necessary conditions for deadlock [2].

** From here on, vectors will be distinguished by the symbol $\wedge$.

Given an execution sequence $\alpha = a_1 a_2 \ldots a_n$, a corresponding state sequence $\sigma = s_0 s_1 s_2 \ldots s_{2n}$ can be defined, each state s_k specifying the amount of available resources and the number of resources units allocated to and requested by each task after the event a_k. Hence, each state s_k is defined by 2n+1 vectors $\hat{v}(k)$, $\hat{P}_i(k)$ and $\hat{Q}_i(k)$, $1 \leqslant k \leqslant n$, where vector $\hat{v}(k)$ specifies the number of available units of each resource type, and vectors $\hat{P}_i(k)$ and $\hat{Q}_i(k)$ specify respectively the number of units of each resource type hold and requested by task T_i after the event a_k. Vectors $\hat{P}_i(k)$ and $\hat{Q}_i(k)$ are defined as follows*:

1. $\hat{P}_i(k) = 0$ if $\exists j : (a_j = \underline{T_i}) \wedge (j \leqslant k)$
2. $\hat{P}_i(k) = \sum_{\forall \ell : T_\ell \in P_i^*} \hat{t}_{\ell i}$ if $\nexists j : (a_j = \overline{T}_i) \wedge (j \leqslant k)$

   where P_i^* is the subset of P_i such that $T_\ell \in P_i^*$ iff $T_\ell \in P_i$ and $\exists h : (a_h = \underline{T_\ell}) \wedge (h \leqslant k)$.
3. $\hat{P}_i(k) = \sum_{\forall \ell : T_\ell \in P_i} \hat{t}_{\ell i} + \hat{q}_i$ if $\exists j : (a_j = \overline{T}_i) \wedge (j \leqslant k)$ and $\nexists h : (a_h = \underline{T_i}) \wedge (h \leqslant k)$.
4. $\hat{Q}_i(k) = \hat{q}_i$ if $\exists h : (a_h = \underline{T_j}) \wedge (h \leqslant k)$ for each $T_j \in P_i$ and $\nexists \ell : (a_\ell = \overline{T}_j) \wedge (\ell \leqslant k)$.
5. $\hat{Q}_i(k) = 0$ otherwise.

In the initial state s_o:

1. $\hat{v}(0) = \hat{R}$, the vector specifying the total amount of system resources.
2. $\hat{P}_i(0) = 0$ for $1 \leqslant i \leqslant n$.
3. $\hat{Q}_i(0) = q_i$ if T_i is an initial task, $\hat{Q}_i(0) = 0$ otherwise.

Given an execution sequence $\alpha = a_1 a_2 \ldots a_k \ldots a_{2n}$, event $a_k = \overline{T}_i$ is <u>allowable</u> if $\hat{Q}_i(k-1) \leqslant \hat{v}(k-1)$. Obviously task terminations are always allowable events. An execution sequence is <u>valid</u> if all initiation events are allowable.
We give now a very intuitive definition of deadlock:

<u>definition 1</u>: Let G be a system of n tasks, $\alpha = a_1 a_2 \ldots a_k$ a partial valid execution sequence, I a set of tasks such that $T_i \in I$ iff $\hat{Q}_i(k) > 0$. State s_k contains a <u>deadlock</u> if there is at least one task $T_i \in I$ such that it is impossible to find a valid partial execution sequence $\alpha' = a_1 a_2 \ldots a_k a_{k+1} \ldots a_p$ defined as follows:

---

* From here on we define a vector $\hat{v} = (v_1, v_2, \ldots, v_j)$ to be a <u>null vector</u> ($\hat{v} = 0$) if $v_i = 0$ for $1 \leqslant i \leqslant j$; a <u>positive vector</u> ($\hat{v} > 0$) is a not null vector such that $v_i > 0$ for $1 \leqslant i \leqslant j$; a <u>negative vector</u> ($\hat{v} < 0$) is a not null vector such that $v_i \leqslant 0$ for $1 \leqslant i \leqslant j$.

1. α is prefix of α'
2. $a_p = \overline{T}_i$

Task T_i is then said to be deadlocked.

Definition 1 says that the system contains a deadlock if we are unable to initiate a task which is ready to be executed, all its predecessors being terminated. Clearly, if a task is deadlocked, all its successors are also blocked, and a whole subgraph cannot be executed.
A deadlock detection procedure based directly on definition 1 is necessarily enumerative. Therefore, we must look for necessary and sufficient conditions which can be more easily verified. Such conditions are stated in the following theorem:

Theorem 1: Let G be a system of n tasks and $\alpha = a_1 a_2 \ldots a_k$ a partial valid execution sequence. Suppose that there is a non empty set D of indices such that for each i in D:

$$\hat{Q}_i(k) \not< \hat{v}(k) + \sum_{j \notin D \vee D^*} \hat{P}_j(k) \qquad (2.2)$$

where D^* is the set of indices such that $\ell \in D^*$ iff T_ℓ is a successor of T_i, for any $i \in D$. Then s_k contains a deadlock and every task T_i, $i \in D$, is deadlocked.

Theorem 1 is an obvious generalization of Denning's definition, but its proof is quite cumbersome and so it is not discussed here (see Appendix). This theorem is interesting mainly because it proves that the definition of deadlock given in [2] can be slightly modified to cover the case of graph structured task systems and gives a simple method to detect deadlock situations. However, our specific goal is avoidance rather than detection, and to avoid deadlock each state must be checked for safeness, according to the following definitions:

Definition 2: Let $\alpha = a_1 a_2 \ldots a_k$ be a partial valid execution sequence: state s_k is safe if there exists at least one valid complete execution sequence having α as a prefix.

In the following sections a further step will be made, looking for sufficient conditions for safeness. We are restricting ourselves to sufficient conditions because we are unable to find necessary and sufficient conditions that are not essentially enumerative.

3. Tree structured task systems

We will consider at first a special case, that of a tree structured task system. This model can represent a situation in which a process, at a particular stage of its activi-

ty, starts the execution on one or more processes, which run independently of each other, starting in turn new sets of processes and so on.
A sufficient condition for safeness can be based on condition (2.2). In fact, suppose that if task T_i can be initiated, then the whole subtree G_i, having T_i as root, can be terminated; this means that there is at least one partial execution sequence, involving the whole set of tasks of subtree G_i, that is valid if the resources granted to T_i become available to its successors. In such a case, if no task asking for resources is deadlocked, then the system is in a safe state. These remarks may be formalized as follows:

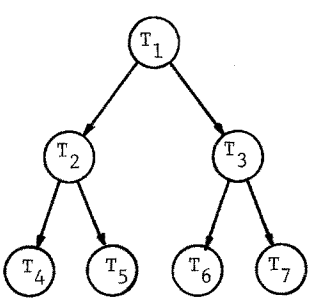
Theorem 2: Let G be a tree structured task system, $\alpha = a_1 a_2 \ldots a_k$ a partial valid execution sequence and I a set of indices such that $i \in I$ iff the immediate predecessor of T_i is terminated but $T_i \notin \alpha$. Each task T_i is the root of a subtree G_i. Let $\hat{M}_i^*$ be a vector representing the minimal amount of resources that must be allocated to subtree G_i to guarantee that all tasks belonging to G_i can be terminated. Let $\hat{Q}_i^*(k) = \hat{M}_i^* - \hat{P}_i(k)$, for each $i \in I$. If no set of indices $D \subset I$ can be found, such that, for each $i \in D$:

$$\hat{Q}_i^* \not\leqslant \hat{v}(k) + \sum_{j \notin D} \hat{P}_j(k) \qquad (3.1)$$

then state s_k is safe.

Proof. If no set D exists satisfying (3.1), then at least one index $i_1 \in I$ exists, such that $\hat{Q}_{i_1}^* \leqslant \hat{v}(k)$ (otherwise a set D = I would satisfy (3.1)). This means that the available resources suffice to complete subtree G_{i_1}. On completion, G_{i_1} release all allocates resources, which are $\hat{P}_{i_1}(k)$ plus all subsequently obtained units up to the maximum $\hat{M}_{i_1}^*$. Therefore, the amount of available resources is now $\hat{v}(k) + \hat{P}_{i_1}(k)$, but, by hypothesis, at least one index $i_2 \in I$ exists such that $\hat{Q}_{i_2}^* \leqslant \hat{v}(k) + \hat{P}_{i_1}(k)$; so subtree G_{i_2} can be completed and so on.

Condition (3.1) is only sufficient, as can be proved by a simple counter-example. Consider the following system:



$\hat{R} = (3,3)$
$\hat{q}_1 = (2,2)$, $\hat{r}_1 = (0,0)$, $\hat{t}_{12} = \hat{t}_{13} = (1,1)$
$\hat{q}_2 = (1,1)$, $\hat{r}_2 = (0,0)$, $\hat{t}_{24} = \hat{t}_{25} = (1,1)$
$\hat{q}_3 = (1,1)$, $\hat{r}_3 = (2,1)$, $\hat{t}_{36} = (0,1)$, $\hat{t}_{37} = (0,0)$
$\hat{q}_4 = (1,0)$, $\hat{r}_4 = (2,1)$
$\hat{q}_5 = (2,1)$, $\hat{r}_5 = (3,2)$
$\hat{q}_6 = (3,1)$, $\hat{r}_6 = (3,2)$
$\hat{q}_7 = (2,2)$, $\hat{r}_7 = (2,2)$

Let $\alpha = \overline{T}_1 \underline{T}_1$ be the partial execution sequence. Then state s_2 is defined by:

1. $\hat{v}(2) = (1,1)$
2. $\hat{P}_2(2) = \hat{P}_3(2) = (1,1)$, $\hat{P}_i(2) = (0,0)$ for $i \notin \{2,3\}$
3. $\hat{Q}_2(2) = \hat{Q}_3(2) = (1,1)$, $\hat{Q}_i(2) = (0,0)$ for $i \notin \{2,3\}$
4. $\hat{M}_2 = \hat{M}_3 = (3,2)$
5. $\hat{Q}_2(2) = \hat{Q}_3(2) = (2,1)$

It can be easily verified that the set of indices $D = \{2,3\}$ satisfies (3,1) but the system is not deadlocked, because the sequence

$$\alpha' = \overline{T}_1 \underline{T}_1 \overline{T}_3 \underline{T}_3 \overline{T}_2 \underline{T}_2 \overline{T}_4 \underline{T}_4 \overline{T}_5 \underline{T}_5 \overline{T}_6 \underline{T}_6 \overline{T}_7 \underline{T}_7$$

is a valid execution sequence.

The central point in theorem 2 is the evaluation of the vectors $\hat{M}^*$'s.

A truly minimal vector is quite difficult to define for two main reasons. First, it is not guaranteed that an unique minimal vector exists; in the above example we could define a different vector $\hat{M}_3^{*\prime} = (2,3)$, which is as minimal as $\hat{M}_3^*$, because if two units of the first type and three of the second type were allocate to subtree G_3 all tasks of G_3 could be completed, and $\hat{M}_3^{*\prime} \not\geq \hat{M}_3^*$ (as well as $\hat{M}_3^* \not\geq \hat{M}_3^{*\prime}$). Secondly, the problem of finding a vector $\hat{M}_i^*$ for subtree G_i is essentially the same as sequencing all the tasks of a tree to verify if, for a given set of system resources, the initial state is safe. To be sure that a set of resources is minimal, we must prove that substracting even one unit of resource we make the initial state unsafe; this test could be performed in a non enumerative way if necessary conditions for safeness were known, but theorem 2 gives us only sufficient conditions. Therefore, to avoid enumerative methods, we will look for a minimal vector $\hat{M}_i \geq \hat{M}_i^*$, trying to define $\hat{M}_i$ as close as possible to $\hat{M}_i^*$. A recursive method to compute $\hat{M}_i$'s can be based on the following theorem:

Theorem 3: Let G a tree structured task system, G_i a subtree having task T_i as root, $\{G_{i_1}, G_{i_2}, \ldots, G_{i_n}\}$ the set of subtrees having as roots the immediate successors of T_i, and T_s the immediate predecessor of T_i.

Let I be the set of indices such that $j \in I$ iff $T_j \in S_i$. Let $\hat{v}_i$ be a vector of resources such that no set of indices $D \subset I$ can be found having the following property:

$$\hat{M}_j - \hat{t}_{ij} \not\leq \hat{v}_i + \sum_{k \notin D} \hat{t}_{ik} \quad \text{for each } j \in D \tag{3.2}$$

Choose $\hat{v}_i$ such that no vector $\hat{v}_i' < \hat{v}_i$ can be found which satisfies the above requirement. Then

$$\hat{M}_i = \hat{v}_i + \hat{t}_{si} + \hat{q}_i - \hat{r}_i \tag{3.3}$$

The proof is trivial and will be omitted.

The computation of $\hat{M}_i$ can be performed only once for every subtree G_i of G, because $\hat{M}_i$ is independent of the specific execution sequence, as stated by (3.3) (in fact, no

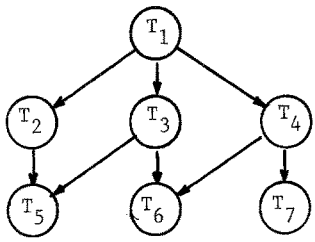
reference is made to any state of the sequence σ). To compute $\hat{M}$'s the best approach is to find the immediate predecessors of terminal tasks, each identifying a subtree, to evaluate the resource requirements of this first set of subtrees, and to go on in this manner reducing step by step the tree to its root.

4. Graph structured task systems

In the case of graph structured task systems theorem 2 cannot be applied because of the complexity of precedence relations; therefore a different approach to safeness verification is needed.

Informally, a criterion for safeness can be stated as follows. Given a task system G, suppose that a partial valid execution sequence α_k has been found, containing the terminations of all tasks at a level greater than j; suppose also that, if all tasks at a level j are terminated, then a valid completion sequence can be found. Then the state s_k is safe if no task at level j is deadlocked. By recursion a general rule to verify safeness can be stated, which implies the possibility of executing the task system level by level.

Consider, for instance, the following task system



and suppose it can be executed level by level, that is the following execution sequence is valid:

$$\alpha^*: \quad \overline{T}_1 T_1 \overline{T}_2 T_2 \overline{T}_3 T_3 \overline{T}_4 T_4 \overline{T}_5 T_5 \overline{T}_6 T_6 \overline{T}_7 T_7$$

Consider now the partial execution sequence $\alpha: \overline{T}_1 T_1 \overline{T}_2 T_2 \overline{T}_5$; the activation of task $\overline{T}_5$ does not lead to an unsafe state only if the amount of available resources after its termination allows us to execute tasks T_3 and T_4.

We will now formalize the above criterion, proving at first the following lemma.

lemma 1: Let G be a system of tasks, $\alpha = a_1 a_2 \ldots a_k$ a partial valid execution sequence, I a set of indices such that $i \in I$ if $\hat{Q}_i(k) > 0$. Suppose $\hat{q}_i - \hat{r}_i \gtrless 0$ for each $i \in I$. Let $I_p \subset I$ be the subset of I such that $i \in I_p$ iff $\hat{q}_i - \hat{r}_i < 0$ and $I_n \subset I$ the subset of I such that $j \in I_n$ iff $\hat{q}_j - \hat{r}_j \geqslant 0$. If no set of indices $D_p \subset I_p$ exists such that, for

each $i \in D_p$:

$$\hat{q}_i \not\leqslant \hat{v}(k) + \sum_{\substack{j \in I_p \\ j \notin D_p}} (\hat{r}_j - \hat{q}_j) \tag{4.1}$$

and no set of indices $D_n \subset I_n$ exists such that, for each $i \in D_n$:

$$\hat{r}_i \not\leqslant \hat{v}(k) + \sum_{j \in I_p} (\hat{r}_j - \hat{q}_j) + \sum_{j \in D_n} (\hat{r}_j - \hat{q}_j) \tag{4.2}$$

then a partial valid completion sequence α_1 exists, such that $\underline{T}_i \in \alpha_1$ for every $i \in I$.

Proof. If no set D_p exists such that (4.1) is satisfied, there is at least one task T_{i_1}, $i_1 \in I_p$, such that $\hat{q}_{i_1} \leqslant \hat{v}(k)$; otherwise a set $D_p \equiv I$ would exist. This means that task T_{i_1} can be initiated with the currently available resources. Let I_{p_1} be the subset of I_p made of all indices $i \in I_p$ except i_1. No set $D_p = I_{p_1}$ exists such that (4.1) is satisfied; therefore there is at least one task T_{i_2}, $i_2 \in I_{p_1}$, such that $q_{i_2} \leqslant v(k) + -q_{i_1} + r_{i_1}$. But the quantity $\hat{v}(k) - \hat{q}_{i_1} + \hat{r}_{i_1}$ represents the amount of available resources after the termination of T_{i_1}, and hence task T_{i_2} can be initiated after the termination of T_{i_1}. Proceeding in this manner it is possible to prove that a valid partial execution sequence $\alpha_2 = \overline{T}_{i_1} \underline{T}_{i_1} \overline{T}_{i_2} \underline{T}_{i_2} \ldots$ exists, such that $\underline{T}_i \in \alpha_2$ for each $i \in I_p$. After the execution of all tasks T_i, $i \in I_p$, the amount of available resources is

$$\hat{v}' = \hat{v}(k) + \sum_{j \in I_p} (\hat{r}_j - \hat{q}_j) > \hat{v}(k)$$

The execution of each task T_i, $i \in I_n$, does not increase the available resources, being $\hat{r}_i - \hat{q}_i \leqslant 0$. If no set D_n exists such that (4.2) is satisfied, there is at least one task T_{ℓ_1} such that

$$\hat{q}_{\ell_1} \leqslant \hat{v}' - \sum_{\substack{j \in I_n \\ j \neq \ell_1}} (\hat{q}_j - \hat{r}_j)$$

otherwise a set $D_n = I_n$ would exist. Therefore task T_{ℓ_1} can be initiated with the resources which are available after the execution of all tasks T_ℓ, $\ell \in I_{n_1}$, being $I_{n_1} = I_n - \{\ell_1\}$. But no set $D_{n_1} \equiv I_{n_1}$ exists, so that at the end a partial valid execution sequence $\alpha_3 = \ldots\ldots \overline{T}_{\ell_2} \underline{T}_{\ell_2} \overline{T}_{\ell_1} \underline{T}_{\ell_1}$ can be found, such that $T_\ell \in \alpha_3$ for each $\ell \in I_n$. Therefore, $\alpha_1 = \alpha_2 \parallel \alpha_3$.

Conditions (4.1) and (4.2) are also necessary if the set of possible sequences α_1 is limited to sequen ces containing no events $\overline{T}_i$ or $\underline{T}_i$ with $i \notin I$. The proof is trivial and will be omitted. Also the following corollary can be trivially proved:

corollary 1. Let G be a system of tasks, $\alpha = a_1 a_2 \ldots a_k$ a partial valid execution se-

quence, I a set of indices such that $i \in I$ iff $\hat{Q}_i(k) > 0$. Suppose that a subset $I' \subset I$ exists such that $\hat{q}_i - \hat{r}_i \not\leqslant 0$ for each $i \in I'$. For each $i \in I'$ substitute $\hat{r}_i$ with a new vector $\hat{r}'_i = (r'_i = (r'_{i1}, r'_{i2}, \ldots, r'_{im})$ such that $r'_{ij} = r_{ij}$ if $q_{ij} \geqslant r_{ij}$, and $r'_{ij} = q_{ij}$ if $\hat{q}_{ij} < \hat{r}_{ij}$. Then lemma 1 can be applied but conditions (4.1) and (4.2) are never necessary.

Suppose now that a task system G can be executed level by level, that is a valid execution sequence $\alpha^* = a^*_1 a^*_2 \ldots a^*_{2n}$ exists such that if $a^*_i = \underline{T}_i$, $a^*_j = \overline{T}'$ and $\ell(T) > \ell(T')$, then $i < j$. Let $\alpha = a_1 a_2 \ldots a_k$ be any valid partial execution sequence, S' a set of integers such that $s \in S'$ iff:

1. $\exists i : \overline{T}_i \in \alpha \qquad \ell(T_i) = s$, and

2. $\exists j : \overline{T}_j \notin \alpha$ and $\ell(T_j) = s$.

and S a set of integers such that $s \in S$ iff $\min\{S'\} \leqslant s \leqslant \max\{S'\}$. For each $s \in S$ let I_s be a set of indices such that $i \in I_s$ iff $\overline{T}_i \notin \alpha$ and $\ell(T_i) = s$. Modify every vector $\hat{r}_i$, $\forall i \in I_s$, $\forall s \in S$, according to the assumptions of corollary 1, if $\hat{q}_i - \hat{r}_i \not\leqslant 0$. Let I_{sp} be the subset of I_s such that $i \in I_{sp}$ iff $\hat{q}_i - \hat{r}_i < 0$, and I_{sn} be the subset of I_s such that $i \in I_{sn}$ iff $\hat{q}_i - \hat{r}_i > 0$. Under these assumptions, the following basic theorem can be proved:

Theorem 4: For every $s \in S$, if no set of indices $D_{sp} \subset I_{sp}$ exists, such that, for each $i \in D_{sp}$:

$$\hat{q}_i \not\leqslant \hat{v}(k) - \sum_{\substack{s' \in S \\ s' > s}} \sum_{t \in I_{s'}} (\hat{q}_t - \hat{r}_t) + \sum_{\substack{j \in I_{sp} \\ j \notin D_{sp}}} (\hat{r}_j - \hat{q}_j) \tag{4.3}$$

and no set of indices $D_{sn} \subset I_{sn}$ exists, such that, for each $i \in D_{sn}$:

$$\hat{r}_i \not\leqslant \hat{v}(k) - \sum_{\substack{s' \in S \\ s' > s}} \sum_{t \in I_{s'}} (\hat{q}_t - \hat{r}_t) + \sum_{j \in I_{sp}} (\hat{r}_j - \hat{q}_j) + \sum_{j \in D_{sn}} (\hat{r}_j - \hat{q}_j) \tag{4.4}$$

then the state s_k is safe.

Proof. Let s_{max}, s_{min} be respectively the maximum and minimum of set S. By assumption, for every $i \in I_{s_{max}}$, if $T_j \prec T_i$, then $\overline{T}_j \in \alpha$; therefore task T_i is ready to be initiated or will be ready after a finite time interval. In any case, the initiation of T_i is constrained only by the availability of resources. But, by lemma 1, if conditions (4.3) and (4.4) are satisfied, a partial valid execution sequence $\alpha_{s_{max}}$ can be found, such that $\underline{T}_i \in \alpha_{s_{max}}$ for every $i \in I_{s_{max}}$. Hence all tasks at level $s_{max} - 1$ will be

ready for initiation after a finite time form event a_k. But, by lemma 1, if conditions (4.3) and (4.4) are satisfied, a partial valid execution sequence $\alpha_{s_{max}-1}$ can be found, such that $T_i \in \alpha_{s_{max}-1}$ for every $i \in I_{s_{max}-1}$, and so on. Therefore, if conditions (4.3) and (4.4) are satisfied, all task at a level $s \geq s_{min}$ can be completed with the currently available resources; let $\alpha_1 = \alpha_{s_{max}} \| \alpha_{s_{max}-1} \| \ldots \| \alpha_{s_{min}}$ be the valid partial completion sequence containing the terminations of all tasks T_i, for $i \in \bigcup_{s \in S} I_s$. Consider now the partial execution sequence α_2 containing all and only all the terminations of all task T such that $\overline{T} \in \alpha$ but $\underline{T} \notin \alpha$; a termination event does not decrease the amount of available resources, and therefore a new valid partial execution sequence α' can be built merging α_1 and α_2. If $\alpha^*_{s_{min}}$ is the prefix of the valid execution sequence α^* such that $\underline{T} \in \alpha^*_{s_{min}}$ for every T such that $\ell(T) \geq s_{min}$ and $\underline{T'} \notin \alpha^*_{s_{min}}$ for every T' such that $\ell(T') < s_{min}$, then the two sequences $\alpha \| \alpha'$ and $\alpha^*_{s_{min}}$ contain exactly the same events; therefore, if α'^* in the completion sequence such that $\alpha^* = \alpha^*_{s_{min}} \| \alpha'^*$, then α'^* is also a valid completion sequence for the partial execution sequence $\alpha \| \alpha'$, that is $\alpha \| \alpha' \| \alpha'^*$ is a valid execution sequence, and consequently the state s_k is safe.

The last problem is to find sets D_p and D_n in lemma 1, D in theorems 1 and 2, and D_{sp} and D_{sn} in theorem 4. This problem is easily solved because:

1. All sets can be searched independently of each other.
2. The search always involves a set of tasks which, upon termination, do not increase (decrease) the amount of available resources; therefore the order of allowable initiation events is not important [2].

Then the simplest algorithm for the detection of sets D_p and D_n in lemma 1 is the following:

Algorithm 1:

Step 1: detection of set D_p

1. Initialize $D_p = I_p$ and $\hat{V} \leftarrow \hat{v}(k)$.
2. Search for any index $i \in I_p$ such that $\hat{q}_i \leq \hat{V}$; if none is found goto 4.
3. $D_p \leftarrow D_p - \{i\}$; $\hat{V} \leftarrow \hat{V} + (\hat{r}_i - \hat{q}_i)$; goto 2.

Step 2: detection of set D_n

4. Initialize $D_n = I_n$ and $\hat{V} \leftarrow \hat{V} + \sum_{j \in I_n} (\hat{r}_j - \hat{q}_j)$.
5. Search for any index $i \in I_n$ such that $\hat{r}_i$ $\hat{V}$; if none is found terminate the algorithm.
6. $D_n \leftarrow D_n - \{i\}$; $\hat{V} \leftarrow \hat{V} + (\hat{q}_i - \hat{r}_i)$; goto 5.

If the number of tasks in $I_p \cup I_n$ is z, the running time of the algorithm is $O(m \cdot z^2)$. Using more sophisticated techniques, it is possible to speed up the algorithm, reaching

a running time of $O(m \cdot z)$ [2]. Sets D_{sp} and D_{sn} in theorem 4, and set D in theorems 1 and 2 can be found in a very similar fashion.

5. Conclusions

The formulation of the deadlock detection and avoidance problem made by Coffman and Denning has been extended here to cover the more general case of a graph structured task system. The proposed approach preserves the basic property of the simpler models developed for independent chains of tasks, i.e. it allows the definition of non-enumerative avoidance algorithms. The conditions on which such algorithms are based are only sufficient, and therefore it can happen that a resource request is not accepted even if it could be granted without entering an unsafe state; however, without this approximation, we are unable to design algorithms which are not essentially enumerative.

APPENDIX

Proof of theorem 1. Suppose that at least one index $i \in D$ exists such that it is possible to find a partial valid execution sequence $\alpha' = a_1 a_2 \ldots a_k a_{k+1} \ldots a_p$, α being a prefix of α' and $a_p = \overline{T}_i$. Let the sequence $\alpha'' = a_{k+1} \ldots a_p$ be a subset of α'. Suppose also that α'' does not contain the initiation of any task T_j, $j \in D$. If this is not the case, and a set of events $\{a_{k_1}, a_{k_2}, \ldots, a_{k_p}\}$ exists such that $a_{k_s} \in \alpha''$ and $a_{k_s} = \overline{T}_{i_s}$ for $i_s \in D$, $1 \leqslant s \leqslant p$, it is always possible to impose new values to p and i such that $a_k = a_{k_1}$, $i = i_1$ and $\alpha'' = a_{k+1} \ldots a_{k_1 - 1}$. Task T_i is initiated, hence $\hat{Q}_i(p-1) \leqslant \hat{v}(p-1)$. Let A be the set of indices such that $j \in A$ iff α'' contains the event $\overline{T}_j$, and B the set of indices such that $j \in B$ iff α'' contains the event $\underline{T}_j$; then:

$$\hat{v}(p-1) = \hat{v}(k) + \sum_{j \in B} \hat{r}_j - \sum_{\ell \in A} \hat{q}_\ell =$$

$$= \hat{v}(k) + \sum_{j \in B'} \hat{r}_j + \sum_{\ell \in B \wedge A} \hat{r} - \sum_{u \in A'} \hat{q}_u - \sum_{s \in B \wedge A} \hat{q}_s =$$

$$= \hat{v}(k) + \sum_{j \in B'} \hat{r}_j - \sum_{u \in A'} \hat{q}_u + \sum_{\ell \in B \wedge A} (\hat{r}_\ell - \hat{q}_\ell)$$

being $B' = B \wedge (B \wedge A)$ and $A' = A \wedge (\overline{B \wedge A})$. If $\hat{P}_{\ell,fin}$ is a vector specifying the total amount of resources held by task T_ℓ after its initiation (and before its termination), then we can write:

$$\hat{v}(p-1) \leqslant \hat{v}(k) + \sum_{j \in B'} \hat{r}_j + \sum_{\ell \in B \wedge A} (\hat{r}_\ell + \hat{P}_\ell(k) + \sum_{s \in B} \hat{t}_{s_\ell} - \hat{P}_{\ell,fin}) =$$

$$= \hat{v}(k) + \sum_{j \in B'} (r_j + \sum_{s \in B \wedge A} \hat{t}_{js}) + \sum_{\ell \in B \wedge A} (\hat{r} + \sum_{s \in B \wedge A} \hat{t}_{s\ell} +$$

$$+ \hat{P}_\ell(k) - \hat{P}_{\ell,fin}) \leqslant$$

$$\leqslant \hat{v}(k) + \sum_{j \in B'} \hat{P}_j(k) + \sum_{\ell \in B \wedge A} \hat{P}_\ell(k) = \hat{v}(k) + \sum_{j \in B} \hat{P}_j(k)$$

Therefore it can be stated that:

$$\hat{Q}_i(k) = \hat{Q}_i(p-1) \leqslant \hat{v}(k) + \sum_{j \in B} \hat{P}_j(k)$$

By definition $B \wedge D = \emptyset$, and consequently $B \wedge D^* = \emptyset$; hence $\hat{Q}_i(k) \quad \hat{v}'k) + \sum_{j \in D \wedge D^*} \hat{P}_j(k)$, which contradicts the theorem.

So far we have proved that (2.2) is a sufficient condition for deadlock detection. We will now prove that it is also necessary. Let D be a set of indices such that $i \in D$ iff no valid partial execution sequence α_i can be found leading to initiation of T_i and having, as a prefix, α (every task T_i, $i \in D$, is deadlocked). Then, in the best case, it is possible to find a partial valid execution sequence having α as a prefix and containing the termination of all tasks T_j, $j \notin D \vee D^*$. Let $\alpha' = a_1 a_2 \ldots a_k a_{k+1} \ldots a_p$ be such a sequence. Being T_i, $\forall i \in D$, a deadlocked task, $\hat{Q}_i(p) \leqslant \hat{v}(p)$, and no task T_j exists such that $\hat{Q}_j(p) \leqslant \hat{v}(p)$, because only tasks T_i, $i \in D$, are asking for resources in state s_p. Therefore set D satisfies condition (2.2). This means that, if state s_k contains a deadlock, a state s_p, $p \geqslant k$, will be necessarily found, in which condition (2.2) is satisfied.

6. References

Coffman, E.G.jr., Elphick, M.J., Shoshani, A. "System Deadlocks", Computing Surveys 2, (1971), 67-78.

Coffman E.G.jr., Denning, P.J. "Operating Systems Theory", Prentice Hall, 1973.

Habermann, A.N. "Prevention of System Deadlock", Comm. ACM 12, (1969), 373-377.

Havender, J.W. "Avoiding Deadlock in Multitasking Systems", IBM Syst. J. 7, (1968), 74-84.

Hebalkar, P.G. "A Graph Model for Analysis of Deadlock Prevention in Systems with Parallel Computation" , Proceed. IFIP Congress (1971), 168-172.

Howard, J.H.jr. "Mixed Solutions for Deadlock Problem", Comm.ACM 16, (1973), 427-430.

Llewellyn, J.A. "The Deadly Embrace - a Finite State Model Approach", Computer Journal 16, (1973), 223-225.