MRM/144

THE UNIVERSITY OF NEWCASTLE UPON TYNE

COMPUTING LABORATORY

Notes of Deadlock Avoidance on the Train Set

by

P. Merlin

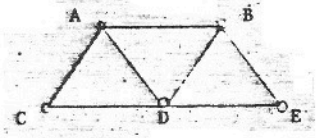
B. Randell

22nd September 1978.

Notes on Deadlock Avoidance on the Train Set

0. Whilst discussing the work that Phil had done on deadlock avoidance (based on buffer assignment techniques) in store and forward message networks, we tried and failed to find some way of using the ideas involved to provide a scheme of deadlock avoidance for our train set. (Each station can hold but one train, i.e. has but one platform (buffer) to allocate.) However, we then embarked on a brief but energetic and intensive exploration of deadlock avoidance based on scheduling constraints, a rough record of which is attempted here.

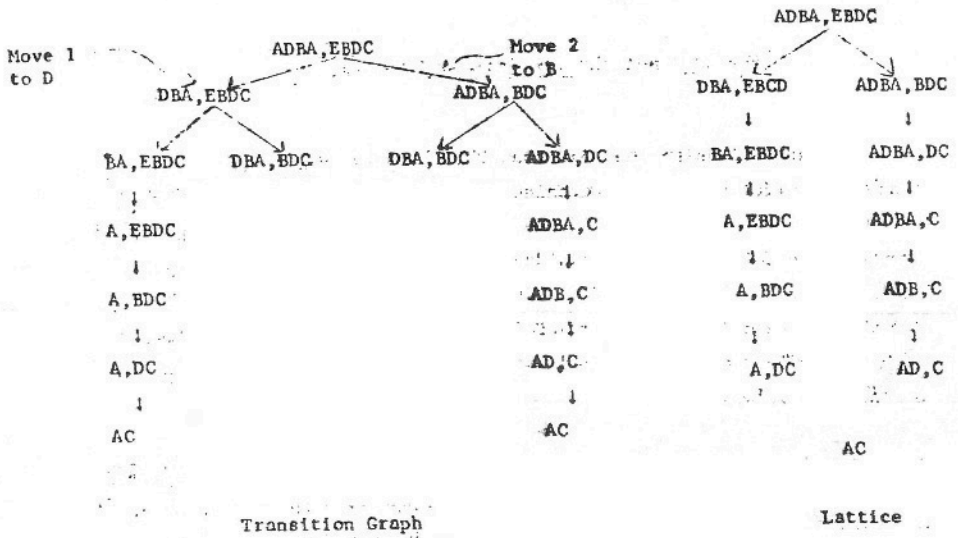
1. The layout is represented by a graph, the places of which represent places where trains can reside ("stations"), the arcs the possible moves. We assume a fixed number of trains, each of which has a finite program to follow, consisting of a directed path through the graph. The train can leave a station only when the station it is to travel to is empty. The problem is to find a sequence of train movements (i.e. an interleaving of the sequences of arc traversals specified by the programs) which allows each program to be completed.



Train 1 : ADBA

Train 2 : EBDC

2. One can represent the (possibly empty) set of all legal interleavings of the steps of the programs (i.e. which do not lead to deadlock) by a transition graph which is a tree whose root is the initial state. Identifying each station by a separate character, then each individual state of the system is represented by a set of character strings, one for each train, indicating where it is and where it still has to go.



A state, to be legal, is such that the leading characters of the strings are all different. A deadlock is represented by a state (other than the final state), from which no legal moves can be made. Such a state and the arc leading to it will be pruned from the transition graph - if in turn this leaves a state which has no arcs leading from it this will also be pruned. Assuming that there is something left of the transition graph after all the pruning, then by also representing each different state only once one gets a lattice, each path through which represents a deadlock-free sequencing of the programs. The bottom of the lattice is the required final state.

3. The original transition graph is big! If the number of trains is N, and the length of each program P, the number of states

$$\sim N^{N+1} \cdot P^2$$

The time to prove that a given system has no solution (i.e. no deadlock-free path) is as above, as might be the time to find such a path, if one is unlucky.

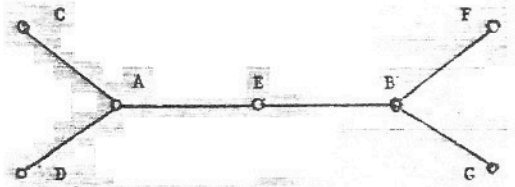
4. However, given the lattice one could have a centralised control scheme for the train which was locally optimum in the sense that it achieved a high degree of parallelism by seeking a path through the lattice, guided

by a desire to keep as many trains running as possible. Train movements represented by arcs along a path would be initiated as soon as the train had reached the starting station, and the destination station was empty (i.e. any train previously in that station had reached its next destination). There might however be a "more parallel" solution, which would require delaying a train whose movement was already possible. This could only be determined if knowledge or estimates of inter-station travel times were available.

5. A given path through the lattice defines, for each station, the sequence of train arrivals which are to be expected. These arrival sequences can be calculated and then used for a distributed control by having each station enforce the appropriate arrival sequence. The resulting set of movements need not follow the original given path - however it will avoid deadlock. (To be exact it will cause a particular one of a class of equivalent deadlock-free paths to be followed.)

The proof involves a Petri-Net, in which there are two sets of places. One set (or more exactly set of sets), represents the sequencing through the arrival sequence, the other the sequencing through the program. This Petri-Net has no loops - and is equivalent to an Occurrence Graph, and hence demonstrates that all programs can be completed.

6. Different paths through the lattice can give rise to different sets of arrival sequences. If, for example, two stations (A and B) each have two different possible arrival sequences then in general it is not satisfactory to have one station enforce the first sequence, and the other the second. An example which demonstrates this is the following:



Train 1 : CAEBF
Train 2 : GBEAD

One deadlock-free path involves stations A and B each enforcing the sequence (1,2)- the other path requires the sequence (2,1). If one station enforced (1,2) and the other (2,1) dealock would not be avoided, rather it would be guaranteed.

7. We have failed, despite much effort, to make progress on the problem of combining such alternative sequencings, where possible, in a way that does not lead to deadlock.

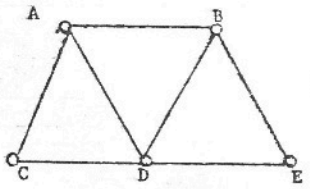
8. A method of cutting down the lattice (but therefore of ignoring some of the deadlock-free paths) is as follows - explore only paths which have at most 'k' trains stopped at stations. (Assuming that inter-station travel times are equal, and that all the trains along a path that could be started are so started.) This makes the lattice size proportional to N^{k+1} - moreover the paths that it finds will be likely to be efficient - i.e. highly parallel (particularly if inter-station journey times are indeed all similar).

9. The above schemes can "easily" (in theory, if not in practice) be generalised to cope with stations which can hold multiple trains. In constructing the lattice, the check is not on whether each string begins with a different letter, but that the largest number of identical letters is within the limits of station capacity.

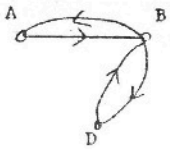
10. Another, more important, extension is to have cyclic rather than linear (finite) programs. Then one does not have a lattice of possible moves, but rather a directed graph. Deadlocks are now represented either by states which have no successors (total deadlocks) or loops which do not allow some trains to progress. Such loops will be ones from which the initial state is not reachable. For the set of programs to be "compatible", the initial state must be in one or more loops of NxP different states (where each program has P steps in its complete cycle). Such cyclic programs thus suffer "tighter" contraints than linear programs, if deadlock is to be prevented or avoided. (We spent only a short time considering this variant of the deadlock problem, hoping to find some way out of the combinatorics inherent in any technique involving searching the systems transition graph. Our theory was that if the constraints" on deadlock-free programs became tighter, it might be easier

to determine whether or not a set of programs met the constraints.)

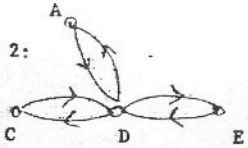
11. A further move in this direction is to have what is effectively a "finite state machine program" specifying the possible movement requests for a train. The states of the machine will be a subset of the states of the graph representing the layout, and the transitions will be directed arcs joining states that are joined by arcs in this graph.



Train 1:



Train 2:

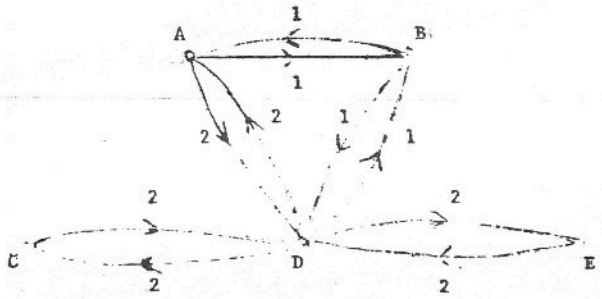


We require that each state of such a finite state machine is reachable from all other states. With such a representation, a train's movements of course are defined solely by its present position, and are independent of how it reached this position. We assume that a train decides arbitrarily which station it wishes to visit next - regardless of whether it will therefore have to wait until such a station becomes free. Moreover, we do not require such decisions to be "fair" - it is possible for a train to remain indefinitely in a loop or loops connecting just a subset of its stations.

12. The first problem of deadlocks with such FSM programs that we tackle is that a deadlock prevention, i.e. that of determining rules which will ensure that a given set of programs can never cause deadlocks to occur, even when no control of the scheduling of train movements is exercised. To tackle this and further problems we represent the set of programs by a "movement graph", consisting of stations connected by labelled directed arcs corresponding to the various train

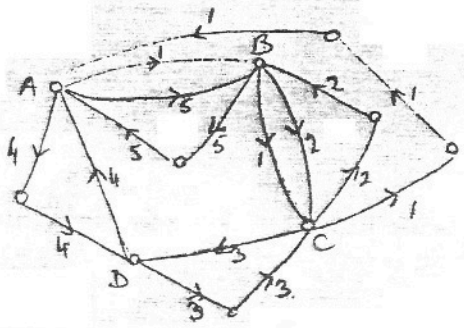
movements. (The labels identify the trains involved.) Clearly only stations which are connected by (undirected) arcs in the layout graph can be connected by (directed) arcs in the movement graph.

The above example produces the movement graph



We can now define a "congestion loop" in such a graph as a directed cycle, of length n, formed from arcs corresponding to n different trains, and a "congestion set" as the set of stations in such a loop.

The above example has no such congestion set.



However, the movement graph shown above has three congestion loops involving the sets of trains (1,2,3,4,), (1,3,4,5) and (2,3,4,5), all corresponding to the same congestion set (A?B?C?D).

13. Deadlock of FSM programs can be prevented, without exercising any control, simply by ensuring that the movement graph formed from the set of programs does not possess any such congestion loops. Lacking any form of control one has to assume that the places connected by such a congestion loop (i.e. the congestion set) can all have a train on them, and that such trains, even if they have alternative paths out of the loop, nevertheless are insisting on trying to reach the next station around the loop. (As always, travel is permitted only to a station which is already empty.)

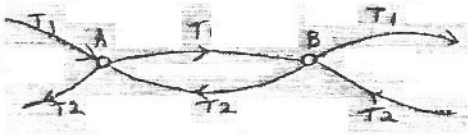
14. The above condition is based on the fact that there is just one train executing each program, so that only one of the arcs labelled with that train's name can be traversed by a train of that type at a given moment, and that station can hold only a single train. If either of these conditions were relaxed - e.g. each station could hold a known maximum number of trains, the definition of a congestion loop would have to be modified appropriately, but the same idea would still be applicable.

15. The class of co-existing FSM programs can be extended by allowing local control. Such local control would be exercised by each station independently, using only knowledge of which trains wish to leave their present (neighbouring) stations to enter it, and which trains have previously reached the station. The control is exercised by preventing such trains from leaving their neighbouring stations.

The actual solution involves:

(1) guarding all entrances to each congestion set,

(2) defining a priori sequences of passages through congestion sets (i.e. locksteppings) which are enforced by the guards at the entrances



Here A and B can be set up, for instance to ensure that T1 and T2 alternate in passing between A and B, or that T1 is allowed five traversals for each one by B, and

(3) limiting the set of programs so as to ensure that trains cannot get stuck elsewhere in their journeys, and so by virtue of the lock-stepped nature of the use of stations in congestion sets, cause other trains to be delayed indefinitely. Thus there must not be any cycles in the programs which are involved in a congestion loop which do not include an arc of the given congestion loop.

16. Yet a further extension of the class of permissible co-existing FSM programs can be allowed if global control is feasible, i.e. if a centralised control system, with knowledge of the entire state of the system, or (equivalently) some means whereby stations can communicate directly with any other station as necessary, is provided.

Again the requirement is to prevent congestion sets from becoming full, and to ensure that trains which are held up from entering such sets are not themselves the cause of deadlocks.

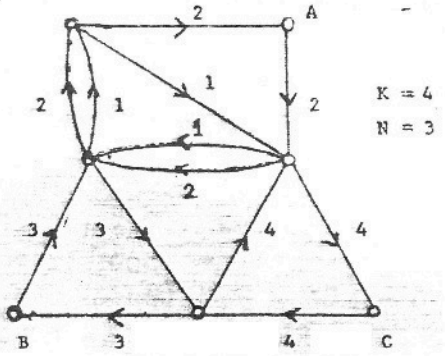
We define contention stations as stations which appear in more than one FSM program, i.e. which at different times could be occupied by different trains. Clearly trains which are at non-contention stations cannot cause any trouble to any other trains.

Consider an n-item congestion set, used by k($k \geq n$) different trains. We must never let more than n-1 of the k trains get into the congestion set at any one time, and must ensure that all the k trains remain able to make progress. To do this we must place some constraints on at least some of the k programs. Specifically it must be the case that there are at least k-n+2 controllable trains, i.e. trains which can be guaranteed to be kept out of the stations without causing trouble - i.e. which have at least one non-contention station which they are guaranteed to reach, where the centralised control scheme can delay them until it is safe for them to leave. The other n-2 trains can then be left uncontrolled, to use the congestion set as they wish - even when they are all in the congestion set there will still be one spare space, so to speak, so that the controllable trains can at least progress through the congestion set one at a time, without suffering deadlock.

To be controllable a train program must not contain any cycles which do not include a non-contention station - without such loops, trains are guaranteed to reach one of their non-contention stations in a finite number of steps.

The centralised control scheme therefore involves, for each congestion set of stations, delaying trains at non-contention stations where necessary in order to ensure that no more than n-1 of the k trains are in contention nodes. Some fair scheduling policy is of course needed to make sure that individual trains are not delayed indefinitely at their non-contention stations.

Example



Train 1 is not controllable. Any of the other trains must be prevented from leaving its non-contention station (A, B or C, respectively) if this would result in all three trains being at contention places.

17. With this last extension and its solution, one is perhaps at last getting near to tackling a practical computing science problem, that of flow control in networks. However honesty requires that this memorandum be left in its present form, i.e. written in terms of trains and stations, rather than packets and nodes.