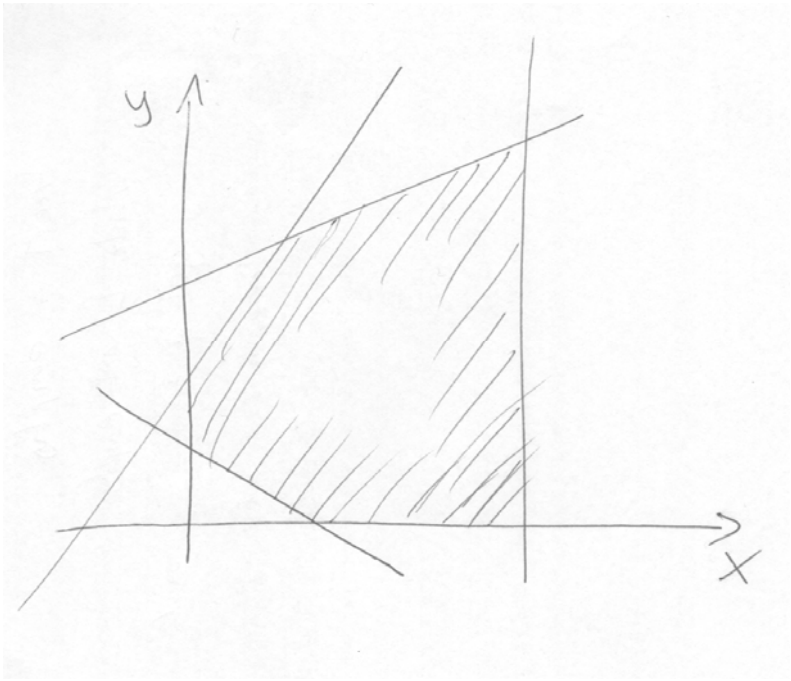
DECO

Discrete Event Control and Optimization

Exam SSY 220, Friday Oct 21, 14:00-18:00, M

Teacher: Martin Fabian, (772) 3716

Time when teacher present: 15:00, 17:00



Solutions and answers should be complete, written in English and be unambiguous and well motivated. In the case of ambiguously formulated exam tasks, the suggested solution with possible assumptions must be motivated. The examiner retains the right to accept or decline the rationality of assumptions and motivations.

In total the exam comprises 25 credits. For the grades 3, 4 and 5, is respectively required 10, 15 and 20 credits.

Solutions will be announced on the course web-page on the first week-day after the exam date. Exam results are announced through Chalmers' administrative routines. The corrected exams are open for review Wednesday Nov 9, 12:30 – 13:30 at the department.

Aids: None.

Task 1. Supervisory Control Theory

A *supervisor* is a control function for a discrete event system. Typically a supervisor is automatically generated from a *plant* describing the possible behavior, and the *specification* describing the allowed behavior of the uncontrolled and controlled system, respectively.

a) Explain the notion of *controllability*, and exemplify what can happen when this property is not fulfilled. (1p)

b) Explain the notion of *non-blocking*, and exemplify what can happen when this property is not fulfilled. (1p)

c) Explain the notion of *minimally restrictive*, and exemplify what can happen when this property is not fulfilled. (1p)

d) Explain the notion of *non-conflict*, and exemplify what can happen when this property is not fulfilled. (1p)

"Controllability" captures the property that the supervisor must always enable ("be ready for") the uncontrollable events that the plant can generate in any state that the supervisor allows the closed-loop system to reach. If the controllability property is not fulfilled, the plant can take the closed-loop system outside the allowed behavior described by the specification.

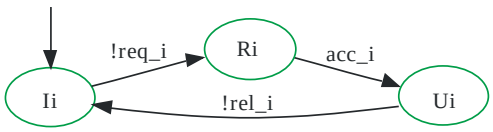
"Non-blocking" describes the property that the supervisor must be such that it always allows at least one of the marked states to be reached. The marked states represent significant "situations", and so at least one of them should always be reachable, otherwise none of these significant "situations" would ever be able to arise. A typical significant "situation" is to reach the initial state.

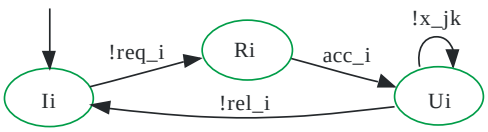
"Minimally restrictive" captures the property that the supervisor should do its job with as little intervention as absolutely possible. Without this property, the degenerate "null" supervisor, which guarantees that nothing bad happens by guaranteeing that nothing at all happens, would be a solution to the synthesis problem.

"Non-conflict" concerns the property that two supervisors supervising the same system should have in relation to each other to guarantee that (if both are individually non-blocking) the closed-loop system of the plant and the two supervisors is non-blocking. If the supervisors are conflicting, they may block the closed-loop system.

Task 2. Discrete Event Specification

The Chalmers student Emilia was asked to give a specification for system of three users using a shared resource, to guarantee that the resource is used by only one user at a time. The plant models consists of three similar automata modeling that each user i ($i \in \{1,2,3\}$) can *request*, get *access* to, and *release* the resource, see below left. The request and release events (*req_i* and *rel_i*, respectively) are uncontrollable, the *access* events (*acc_i*) are controllable. Emilia added to the plant models uncontrollable events *x_jk* (with $j,k \in \{1,2,3\}$, $j \neq k$ and *x_jk* = *x_kj*), so that $x\_jk \in \Sigma_{Ui}$ whenever $j = i$ or $k = i$. These events were self-looped at the *Ui* states, see below right.





Then Emilia added three specifications *Sp_jk* such that each only consisted of a single marked state and had a single uncontrollable event in its alphabet, the *x_jk* event, but no transitions.

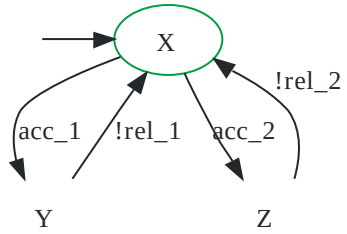
a) Explain in detail why Emilia added those events and gave those specifications. Why would this guarantee that the resource is used by only one user at a time? (3p)

b) Give an alternative formulation of a specification to guarantee that the resource is used by only one user at a time, not involving any extra events, and compare that to Emilias solution. (2p)

Each user model will have two self-looped uc-events at its Ui state. Through synch comp those self-loops will survive only in the states where two users simultaneously use the shared resource. Emilias specifications, that always forbid the uncontrollable x_jk events, generates a controllability problem exactly at those where the x_jk events are self-looped, and hence the synthesis algorithm will remove those states and all possible uc-paths to them. In this way we are guaranteed that that the resource is used by only one user at a time.

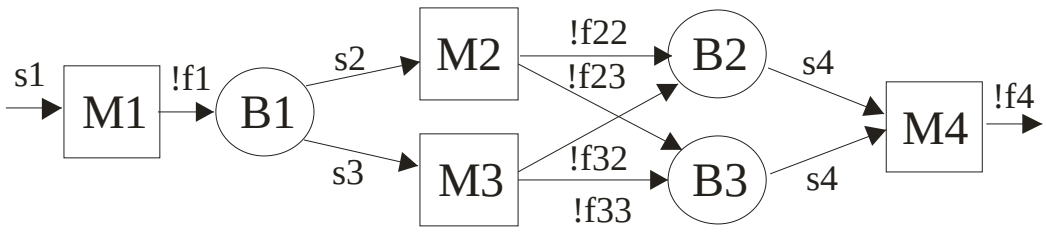
Emilias spec cleverly promotes modular synthesis; a spec is picked, it has a single uc-event in common with two (modified) plants which are then picked out for synthesis, and a modular supervisor is generated.

There are several different ways to specify the same requirement, but all of them require more work. The obvious way is to pair-wise synchronize the (un-modified) plants and to manually remove (or forbid) the state where the two users simultaneously use the resource. Another way is to add three-state specs that pair-wise mutually exclude the acc_i events until the rel_i event has been seen, see below where this is shown for users 1 and 2.



Task 3. Supervisor Synthesis

A small manufacturing system consists of 4 machines with buffers in-between, see below. Machine M1 feeds buffer B1, which in turn holds parts for both M2 and M3. These machines, M2 and M3, feed either B2 or B3. Machine M4 takes parts from both B2 and B3.

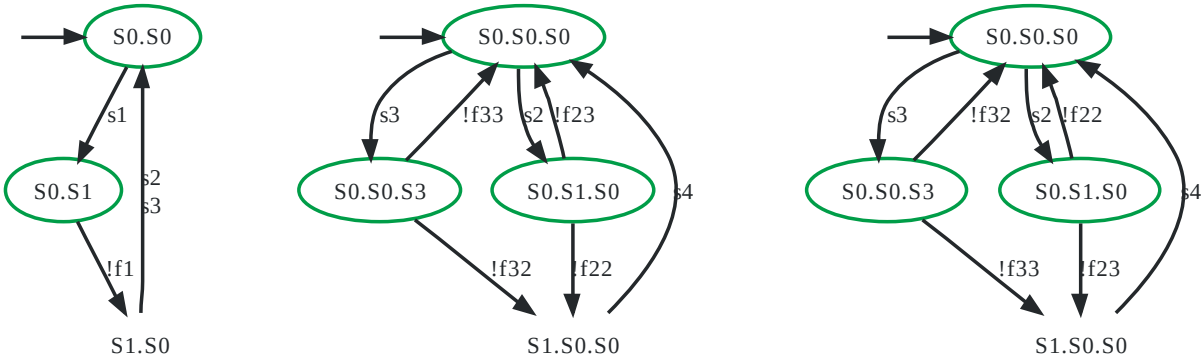


All devices are modeled as two-state automata, where the buffers (acting as specification) have their initial state marked, while the machines have both states marked. The capacity of each buffer is one. The *s*-events (see above) are controllable while the *f*-events are uncontrollable.

a) Model the system and the specification, and synthesize a modular supervisor. (4p)

b) Is the modular supervisor non-blocking? Motivate. (1p)

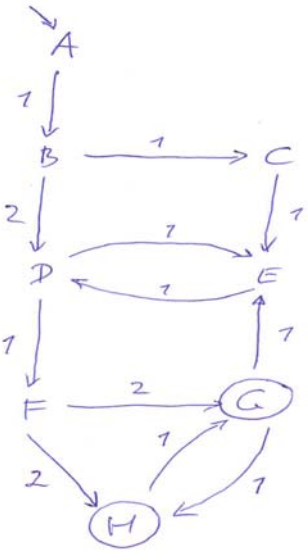
The models of the machines and buffers are rather straightforward. The supervisors are synthesized from B1||M1, B2||M2||M3, and B3||M2||M3, respectively. This gives the following supervisors.



The controlled system is unfortunately blocking. This we can see if we regard the supervisors for B2||M2||M3 and B3||M2||M3. They agree on, for instance, s3 but then one wants to take f33 to a marked state whereas the other wants to take f33 to an unmarked state. This is a classic case of conflict; s3.f33 is a prefix of a marked string in both automata, but it is not a prefix of a common marked string.

Task 4. Discrete Optimization

Below is a weighted directed graph, where the weights are costs noted on the arcs. The initial node is A, and there are two alternative goal nodes G and H.



a) Run Dijkstra's algorithm on the graph to find the goal node with the lowest-cost path from the initial node. Make sure to at each step denote which node is currently examined and which nodes are in the list to possibly be examined. (4p)

b) Describe the main difference between Dijkstra's algorithm and A*. (1p)

Diksttra's algo runs on the above graph as

A[0,-]: B[1,A]:

B[1,A]: C[2,B]: D[3,B]:

C[2,B]: D[3,B]: E[3,C]:

D[3,B]: E[3,C]: F[4,D]:

E[3,C]: F[4,D]:

F[4,D]: G[6,F]: H[6,F]:

G[6,F]: H[6,F]:

And the optimal path is thus

G[6,F]: F[4,D]: D[3,B]: B[1,A]: A[0,-]:

Note that also H could be considered the node with least cost, since it has the same cost as G. Whether G or H is found, depends on the internal storage of the nodes; at F, either G or H could be picked out to be examined, either choice would be correct.

Dijkstras algo is guided by the cost to a certain node. A is guided by both the cost of getting to a node* n *(*g(n)*) plus an estimate of getting from* n *to the goal (*h(n)*). Dijkstra's algorithm is principally the A* algorithm with h(n)=0, that is, it does not use any estimate of the "future" as guide, it merely looks at the "past". Naturally, h(n)=0 is an estimate that never over-estimates the true value, and it guarantees monotonicity.*

For both algorithms, the node weights must be non-negative.

Task 5. Linear and Integer Programming Theory

We have looked at Linear Programming (LP) problems, Integer Programming (IP) problems, and Mixed Integer Linear Programming (MILP) problems.

a) What is the meaning of an *LP-relaxation* of a MILP or IP problem? Why is it useful? (2p)

b) Describe how the LP-relaxation could be used to solve an IP/MILP problem. Be sure to describe how we know when the optimal solution has been found. (3p)

c) What is the importance of the *A*-matrix being *totally uni-modular* for an IP problem? (1p)

An LP-relaxation of an IP/MILP problem is what we get when we disregard the requirement that some (or all) variables should take on integer values. This is essentially is a less constrained problem than the original one, since it has less requirements on the variables, and hence the solution to the LP_relaxed problem is a bound on the true solution. The solution to the full problem can never be better than the solution to the relaxed problem.

It is known that if the solution to the relaxed problem happens to satisfy all constraints of the non-relaxed original problem, then the solution to the relaxed problem is the solution to the original problem. Thus, we can iteratively use solutions relaxed problems to eventually end up with a solution to the original problem. We solve the LP-relaxation, and know that this solution is a bound on the true solution. Then we add constraints to the integer variables to bound them to integer values, and solve new LP-relaxations. We can do this again and again until we eventually end up with a solution where all integer variables take on integer values, and we are done. During this process we know that each relaxation poses a bound on the solution where the integer variables are really integer, and so we can employ a "branch and bound" technique to eventually arrive at the true solution. We know that we have found the true solution when we have a solution that satisfies all requirements, and we have no unexplored relaxations with a better solution value.

The importance of a totally uni-modular A-matrix comes from the fact that for such a matrix, every non-null square sub-matrix is integer, and so if also the b vector is integer, all solutions to such a problem will be integer. Therefore we know that with a totally uni-modular A-matrix and integer b-vector, the LP-relaxation will give us the solution to the true IP problem.
