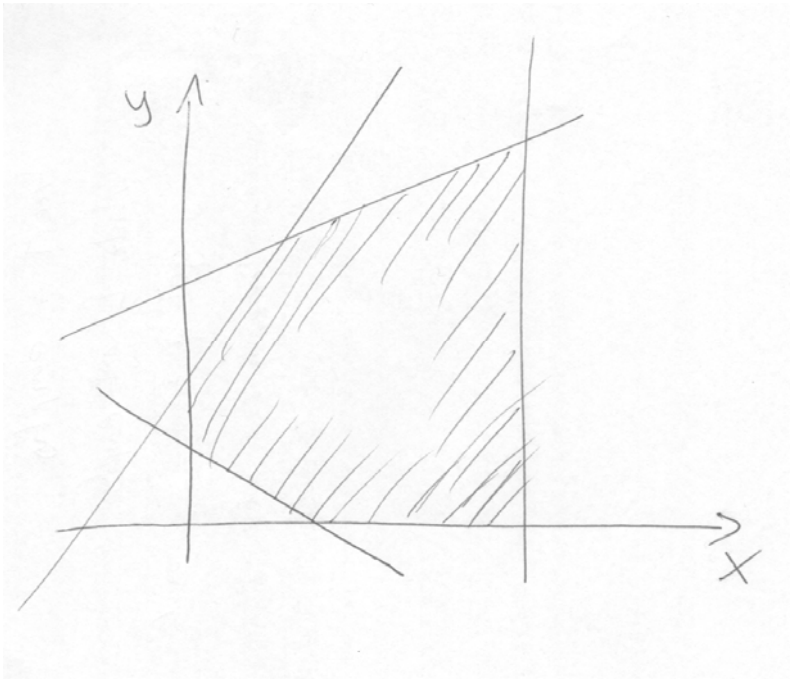
DECO

Discrete Event Control and Optimization

Exam SSY 220, Friday Aug 19, 08:30-12:30, M

Teacher: Martin Fabian, (772) 3716

Time when teacher present: 09:30, 11:30



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 Aug 31, 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.

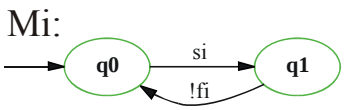
"Non-blocking" describes the property that the supervisor must be such that it always allows at least one of the marked states to be reached. The marked states represent significant "situations", and so at least one of them should always be reachable, otherwise none of these significant "situations" 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. Supervisor Synthesis

Two machines, M1 and M2, are connected in series with a buffer in-between, see below left. The machines are modeled as the simple automaton to the right. The events *fi* ($i = 1, 2$) are un-controllable. The *f1* event represents that M1 is finished and puts a product in the buffer, while the *s2* event represents that M2 takes a product from the buffer. The *s1* and *f2* events represent that M1 gets a product from the outside, and that M2 deposits a product to the outside, respectively.



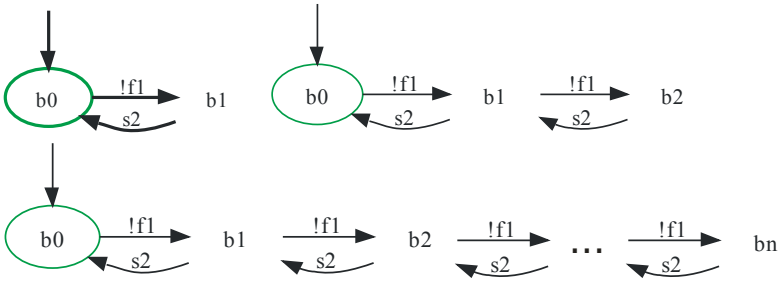
a) Give a specification that guarantees that *at most one product* is in the buffer at all times, and that the system can always return to the initial state, with no products in either of the machines or the buffer. (1p)

b) Give a specification that guarantees that *at most two products* are in the buffer at all times, and that the system can always return to the initial state, with no products in either of the machines or the buffer. (1p)

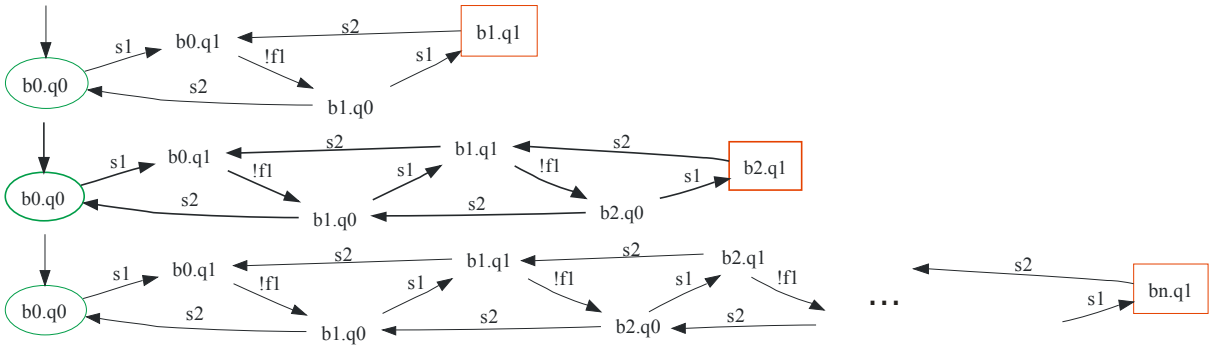
c) Give a specification that guarantees that *at most n products* (with *n* an arbitrary integer larger than zero) are in the buffer at all times, and that the system can always return to the initial state, with no products in either of the machines or the buffer. (1p)

d) From the specification given in c), and the given plant models, synthesize a supervisor. (3p)

The specifications look like this:



Now we can observe that these specs share uc-events only with M1*, so we can do the synthesis with only* M1*, and then reason about whether the result will be non-blocking when we involve also* M2*. Synchronizing the specs with* M1 *gives the following, where the squared states are forbidden, since* M1 *could do the uc-event f1 there, but the spec does not agree.*



As we can see, only controllable events lead to the forbidden states, and thus we can simply remove the states and their transitions, and the result will be controllable. Will it also be non-blocking when M2 *comes in to play? The answer is yes, simply because* M2 *can always execute its s2 event, in the worst we will have to wait for it to execute its "free" f2 event, but always eventually.*

Note one additional thing. As the plants and specs are given, all states with a b0-component would be marked in the synchronization. However, we altered this above to better conform to the intent of the requirement given in the text. This is an insignificant detail (in this case).

Task 3. Discrete Optimization

A* is a branch-and-bound algorithm that uses both the cost $g(n)$ to get to a node n, and an estimate $h(n)$ to get from n to the goal, to guide the search. A* is similar to Dijkstra's algorithm, but they are not equal.

a) When are A* and Dijkstra's algorithms equivalent? (1p)

b) Assume we have two different estimate functions $h_1(\cdot)$ and $h_2(\cdot)$, that both guarantee that the optimal path is found, and where for all n, $h_1(n) \leq h_2(n)$. Describe how A* behaves with the two respective estimate functions related to each other. (1p)

c) Assume that the estimate function $h(n)$ is always equal to the true cost $h^*(n)$ from any node n to the goal. Describe how A* behaves then. (1p)

d) Assume that $h(n)$ is at some nodes larger than the true cost $h^*(n)$. Describe how A* behaves then. (1p)

e) Suggest a realistic estimate, that guarantees that the optimal path will be found, to use when path-planning a production system where a number of jobs concurrently share a number of resources. (1p)

A is guided by both the cost of getting to a node* n *(*g(n)*) and an estimate of getting from* n *to the goal (*h(n)*). To guarantee that the optimal path to the goal is found,*

- h(n) *must not over-estimate the true value* h*(n)*, that is* h(n) <= h*(n)*, and*
- h(n) *must me monotonic, that is, for two nodes* n_1 *and* n_2*, such that* n_2 *is reachable from* n_1 with weight $w(n_1,n_2)$, $h(n_1) <= w(n_1,n_2) + h(n_2)$.

For a system with a number of jobs routed through a number of resources, one such estimate comes from regarding each job as running by itself through its needed resources. Of course, the time to finish a single job must be shorter than finishing several simultaneous jobs competing for the resources.

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.

The tighter (closer to the true value) the estimate, the less nodes are investigated by A, so that when* $h_1(n) \leq h_2(n)$*, then A* examines less nodes with* $h_2(n)$*. If the estimate is so tight that it is equal to the true value, A* will go directly to the goal without examining any other nodes than those along that path. If the estimate over-estimates (that is, is larger than the true value) at some node (that is examined) then A* is not guaranteed to find the optimal path.*

Task 4. Linear Programming

We have formulated LP-problems in a "standard form", looking like

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Az = b \\ & x \geq 0, x \in \mathbb{R} \end{aligned}$$

Regard the following LP-formulation

$$\begin{aligned} \max \quad & -x_1 - x_2 - x_3 \\ \text{s.t.} \quad & x_1 + x_2 \geq 4 \\ & -2x_2 - x_3 \leq 4 \\ & -x_1 - x_3 \leq -4 \\ & x_i \geq 0 \text{ and } x_i \in \mathbb{R} \end{aligned}$$

a) Formulate this as an LP-problem in the standard form given above. (2p)

b) Solve it. (3p)

c) Is the feasible set bounded or unbounded? (1p)

We know that max -f = min f, so we rewrite the objective as $-(-x_1 - x_2 - x_3) = x_1 + x_2 + x_3$.

We need also to include slack and surplus variables to get equalities. For convenience we also rewrite b *to have only non-negatives. So we get:*

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 \\ \text{s.t.} \quad & x_1 + x_2 - s_1 = 4 \\ & -2x_2 - x_3 + s_2 = 4 \\ & x_1 + x_3 - s_3 = 4 \\ & x_i, s_i \geq 0 \text{ and } x_i, s_i \in \mathbb{R} \end{aligned}$$

Writing this in matrix form is trivial.

Note that the constraint $-x_1 - x_3 \leq -4$ *is equivalent to* $x_1 + x_3 \geq 4$. *This constraint together with the constraint* $x_1 + x_2 \geq 4$ *means that the feasible set does not contain (0,0,0).*

The optimal value of the objective function is 4, which occurs when $x_1 = 4$, $x_2 = 0$, $x_3 = 0$.

Note that $-2x_2-x_3 = -(2x_2+x_3)$ *is always negative (when the* x_i *are non-negative) and so can be removed from the problem formulation. So we can actually solve the much simpler problem*

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 \\ \text{s.t.} \quad & x_1 + x_2 \geq 4 \\ & x_1 + x_3 \geq 4 \\ & x_i \geq 0 \text{ and } x_i \in \mathbb{R} \end{aligned}$$

which quite obviously has the above solution. Clearly, the feasible set is unbounded.

Task 5. Linear and Integer Programming Theory

Regard the following four optimization problems:

$$z_1^* = \begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq b \\ & 0 \leq x \leq 1 \end{array} \qquad z_2^* = \begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq b \\ & 0 \leq x \end{array}$$

$$z_3^* = \begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq b \\ & 0 \leq x, \text{integer} \end{array} \qquad z_4^* = \begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \in \{0,1\} \end{array}$$

The z_i^* (for $i \in \{1,2,3,4\}$) are the different optimal objective function values.

a) Determine the relative relationship (in magnitude) between the z_i^* (for $i \in \{1,2,3,4\}$). Note that it may not be possible to relate all of them to each other. (2p)

b) Assume that A is totally uni-modular and that b is integer, then do the same as in a). (2p)

The least constrained problem is the 2^{nd} *one, so that* z_2^* *has potentially the largest value. The most constrained problem is the* 4^{th} *one, so this has potentially the smallest value. As for the* 1^{st} *and* 3^{rd} *problems, we cannot say much about their relative magnitudes, except that they fall in-between the solutions to the* 2^{nd} *and* 4^{th} *problems. Thus we have*

$$z_4^* \leq \begin{Bmatrix} z_1^* \\ z_3^* \end{Bmatrix} \leq z_2^*$$

When A *is totally uni-modular and* b *is integer, the solutions to the* 1^{st} *and* 2^{nd} *problems are integer and so must be the same as the solutions to the* 4^{th} *and* 3^{rd} *problems, respectively. Thus we have that* $z_4^* = z_1^* \leq z_2^* = z_3^*$.