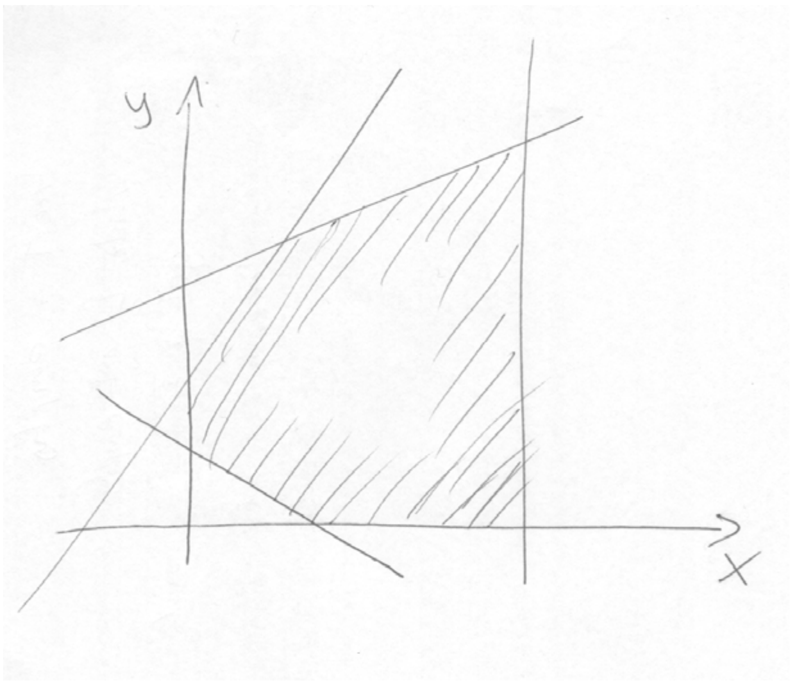
DECO

Discrete Event Control and Optimization

Exam SSY 220, Friday Jan 13, 08:30-12:30, V

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.

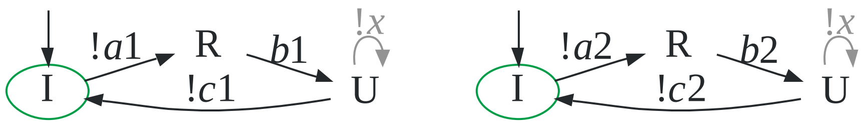
In total the exam comprises 25 credits. For the grades 3, 4 and 5, is respectively required 10, 15 and 20 credits.

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 seven work days after the exam, 12:30 – 13:30 at the department.

Aids: None.

Task 1. Supervisor Specification

Emil made a model (see below) of a system with two users sharing a common resource; this could model a system with two computers sharing a printer. Each user first requests access to the resource (the a_1 and a_2 events), then waits to get access approved (b_1, b_2), uses the resource in the respective *U* state, and then releases the resource (c_1, c_2). The b_i events were controllable, and the others uncontrollable.



Emilia noticed that the original model allowed the resource to be used by the two users simultaneously, which was no-good, of course. So she added the uncontrollable *x* event as a self-loop on the respective *U* states, and then she added to the system a specification automaton *Sp*, with only a single marked initial state, no transitions, and only the uncontrollable event *x* in its alphabet, that is, $Sp := \left\langle \{q_0\}, \{x\}, q_o, \varnothing, \{q_0\} \right\rangle$. Synthesizing a supervisor from this would yield a controllable and non-blocking system that avoided the simultaneous use of the shared resource, she claimed.

Explain in words why her claim is true. (2p)

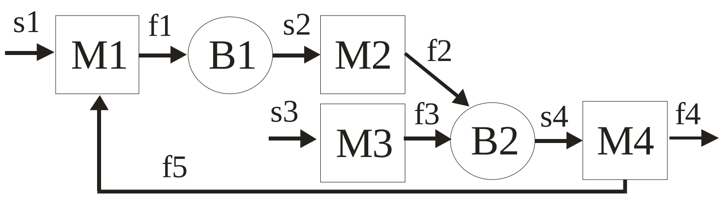
In the composition User$_1$||User$_2$ *the x event will survive only in the* UU *state. Thus,* Sp *effectively forbids this state, since it always forbids the* x *event. Thus, the synthesis must remove the* UU *state to make the composed system (plant and spec) controllable.*

Will it be non-blocking? Yes, since in the RR *state, the system can only do either* b_1 *followed by* c_1*, or* b_2 *followed by* c_2, and thus the system can always progress to reach the marked initial state. Hence, it is non-blocking.

Task 2. Supervisor Synthesis

A manufacturing system consists of 4 machines connected with buffers in-between them, see below. Each machine can hold at most one product. The last machine, M4, either feeds its output back to M1 or moves its work piece out of the system. Thus, M1 gets its input either from outside the system (s_1), or from M4 (f_5). The machines M1, M2 and M3 feed their output into the respective buffers. M3 gets its input from outside the system (s_3). The events f_i ($i = 1 \ldots 5$) are un-controllable, the s_i events are controllable.

The specification is to have *at most one* product in each buffer at all times. We want the system to be able to accept and process products again and again.



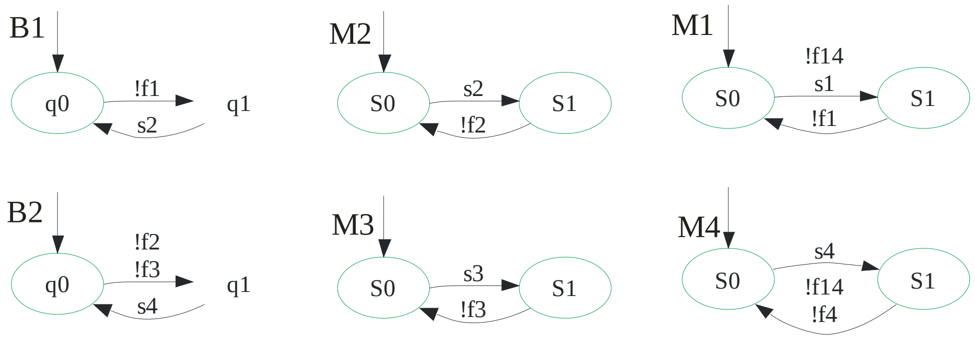
a) Model the plant components (Mi) and the specifications as two-state automata, one automaton for each machine and one automaton for each specification. (2p)

b) Calculate a modular supervisor. (3p)

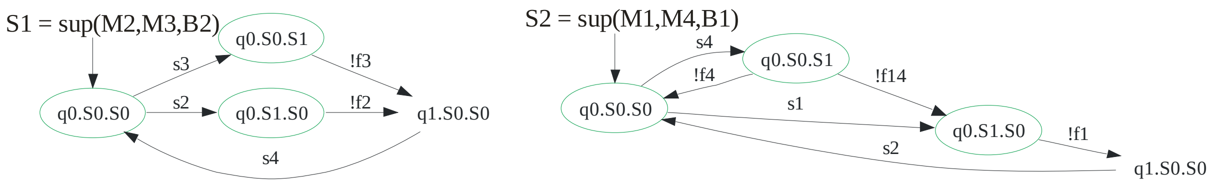
c) Is the modular supervisor resulting from b) minimally restrictive? Motivate. (1p)

d) Is the modular supervisor resulting from b) non-blocking? Motivate. (2p)

The two-state models are fairly straightforward. The specification is, of course the same as the models for the buffers.



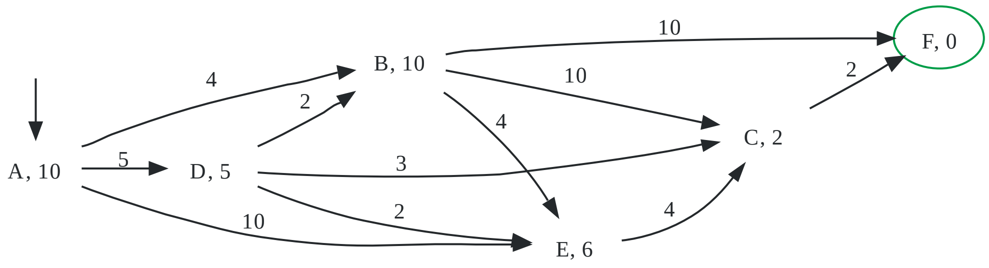
Calculating supervisors we do for B1, M1, M4 and B2, M2, M3 respectively, which gives us the following modular supervisor.



Is this supervisor minimally restrictive? Yes it is, since we included M4 in the synthesis for B1. If we do not include M1, then the result will be null, which is definitely not minimally restrictive.

Is this supervisor non-blocking? Well, we could synchronize both and check. This would result in a 15 state system, so it is doable. But, we can also reason in this way. The two supervisors overlap in s2 and s4, so should there be any blocking problems, it would concern these two events; can we get to a state where one supervisor wants to only do one of these events, while the other supervisor only agrees on the other event? And indeed, after the string s1.f1.s3.f3, the supervisor S1 wants to do an s4, which S2 does not agree on. S2, on the other hand, wants to do an s2 that S1 does not agree on. This is conflict, *and it means that the supervisors are blocking.*

Task 3. Discrete Optimization



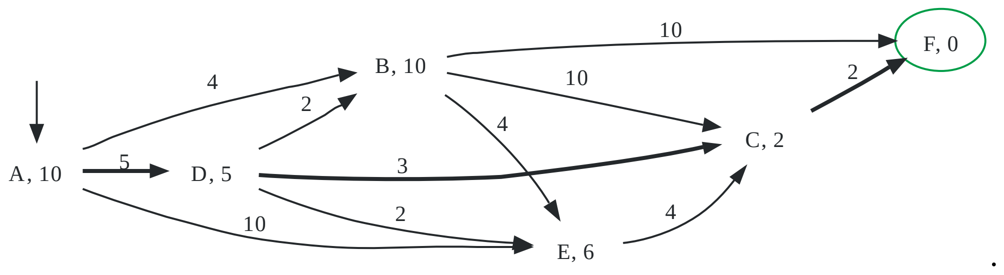
Above is given a graph, with weights on the edges, and for each node an estimate of the cost to reach the goal node *F*.

a) Using Dijkstra’s algorithm, find the least cost path through the graph. (2p)

b) Using the A* algorithm, find the least cost path through the graph. (3p)

In both cases, show on each iteration which node is taken out from and put in to the queue, and also what the queue looks like.

The optimal path is marked by thick lines. It is the same for both algorithms, of course.



The workings of Dijkstra's algorithm is shown to the left, below, and A is to the right.*

Dijkstra's Algorithm		*A**	
A[0,-]	B[4,A] D[5,A] E[10,A]	A[0,10,-]	D[5,5,A] B[4,10,A] E[10,6,A]
B[4,A]	D[5,A] C[14,B] E[8,B] F[14,B]	D[5,5,A]	C[8,2,D] B[4,10,A] E[7,6,D]
D[5,A]	E[7,D] F[14,B] C[8,D]	C[8,2,D]	F[10,0,C] B[4,10,A] E[7,6,D]
E[7,D]	C[8,D] F[14,B]	F[10,0,C]	E[7,6,D] B[4,10,A]
C[8,D]	F[10,C]		
F[10,C]			

The first element within the brackets is the current cost of the node, the last element is the current parent, and the middle element is the estimate.

Note that the given estimates are actually the true optimal values from the node to the goal. Naturally, such estimates fulfill the requirements we place on estimates, monotonic and not over-estimating, and this example also shows that the tighter the estimate, the better A performs. Here it goes straight for the goal, never diverging from the optimal path (which Dikstra's algo does)..*

Task 4. Linear Programming

A company producing plastic part wants to optimize the earnings on its production. Three different plastic parts, *A*, *B*, and *C*, are produced using two machines, *M1*, and *M2*. The time it takes for each machine to process the respective parts, and the total available times are given in the table below.

	M1	*M2*
A	*2*	*3*
B	*1*	*2*
C	*2*	*2*
Total available time	*120*	*150*

The marketing department demands that at least 20 pieces of *A* are manufactured. The advance on each part is €50, €40 and €60, respectively, and the total profit is to be maximized.

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

b) Solve it. (3p)

c) Will there be any time left in any machine? (1p)

Letting x_A etc denote the produced amount of the respective product, we can formulate the problem as:

$$\begin{aligned} \max\quad & 50x_A + 40x_B + 60x_C \\ \text{s.t.}\quad & 2x_A + x_B + 2x_C + s_1 = 120 \\ & 3x_A + 2x_B + 2x_C + s_2 = 150 \\ & x_A - s_3 = 20 \\ & x_i, s_j \geq 0 \end{aligned}$$

Solving this gives the optimal solution €3500 when x_A = 20, x_B = 10, and x_C = 35. At this optimum the slack and surplus variables are all zero, so no processing time will remain in the machines.

Note two things. The variables represent number of produced products, and must hence be integer. Also, the "standard form" is sometimes interpreted as that the objective function should be minimized; this is a trivial and uninteresting transformation.

Task 5. Linear and Integer Programming Theory

Regard the following four optimization problems:

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

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

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 1st one, so that z_1^* has potentially the largest value. The most constrained problem is the 2nd one, so this has potentially the smallest value. As for the 3rd and 4th problems, we cannot say much about their relative magnitudes, except that they fall in-between the solutions to the 1st and 2nd problems. Thus we have

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

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