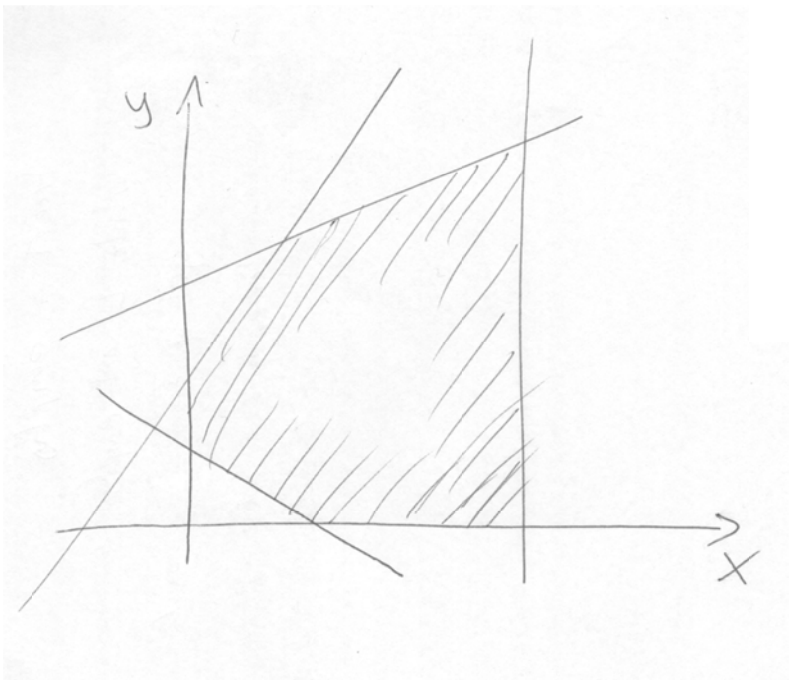
DECO

Discrete Event Control and Optimization

Exam SSY 220, Saturday, May 24, 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.

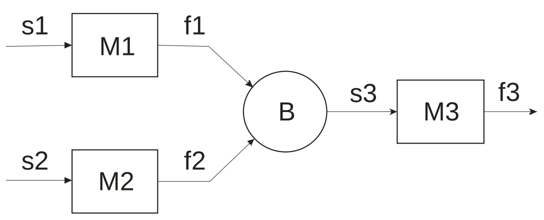
In total the exam comprises 25 credits. For the grades 3, 4 and 5, is respectively required 10, 15 and 20 credits.

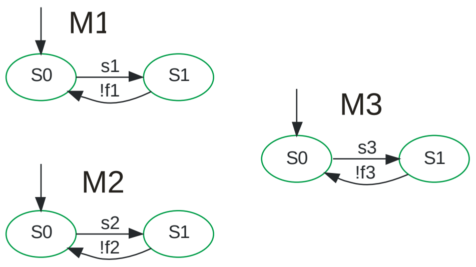
Solutions will be announced on the course web-page on the first week-day after the exam date. Exam results are announced through Chalmers' administrative routines. The corrected exams are open for review seven work days after the exam, 12:30 – 13:30 at the department.

Aids: None.

Task 1. Abstractions for Compositional Methods, and Modular Synthesis

We have a small manufacturing system as shown below. M1, M2 and M3 are machines, and B is a buffer. The plant can be modeled by the simple automata given to the right below. The s_i events (i = 1,2,3) are controllable, while the f_i events are uncontrollable.





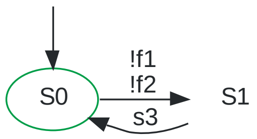
a) Give a specification for the buffer to hold one work piece, never to over- or underflow, and to always end up empty. (2p)

b) Abstract the system as much as possible for compositional nonblocking verification. (2p)

c) Abstract the system as much as possible for compositional supervisor synthesis. (3p)

d) Using modular synthesis, calculate a supervisor for the non-abstracted system. (3p)

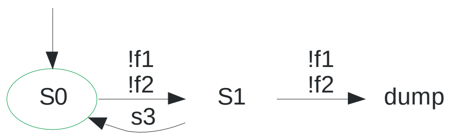
The specification B can look like this:



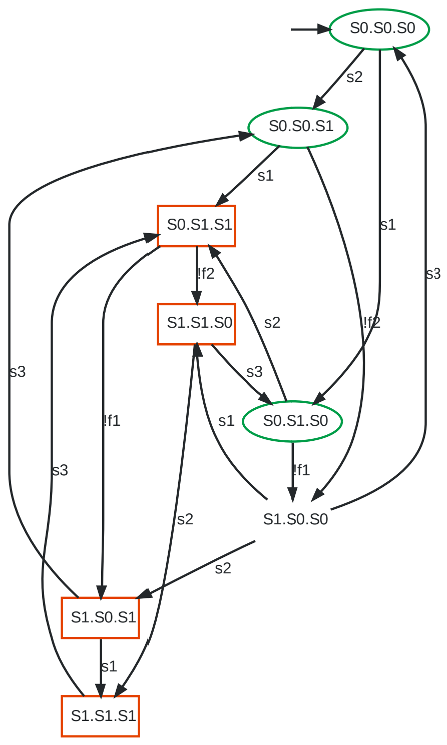
We can start by abstracting M3, noting that !f3 is a local uncontrollable event. This abstraction is valid both for verification and synthesis, and it makes M3 self-loop only so it can be ignored.

For compositional verification, we can merge states that are connected by local events, whether those events are uncontrollable or not, so M1, M2, and B can all be merged into self-loop only automata with their respective single states marked. Thus, the system is obviously nonblocking.

For compositional synthesis, we cannot merge states connected by local controllable events. However, for synthesis we need to "plantify" the specification, which looks like this:

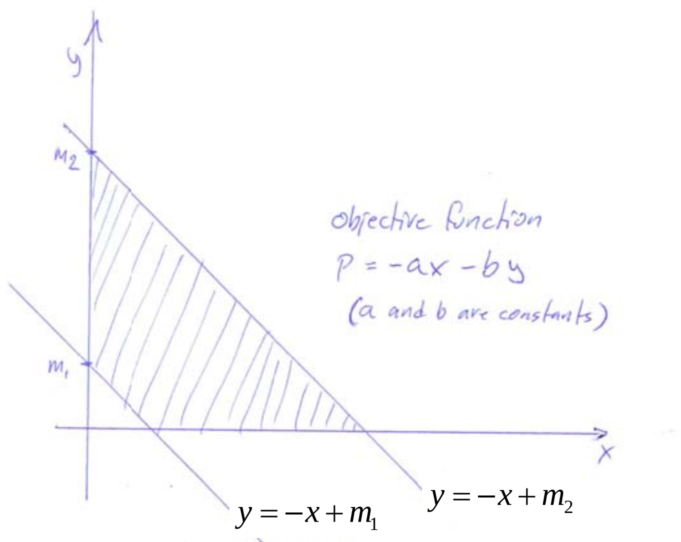


Modular synthesis is straightforwardly done by only considering M1, M2 and B. The result looks like this (the forbidden states can be removed, of course):



Task 2. Linear Programming

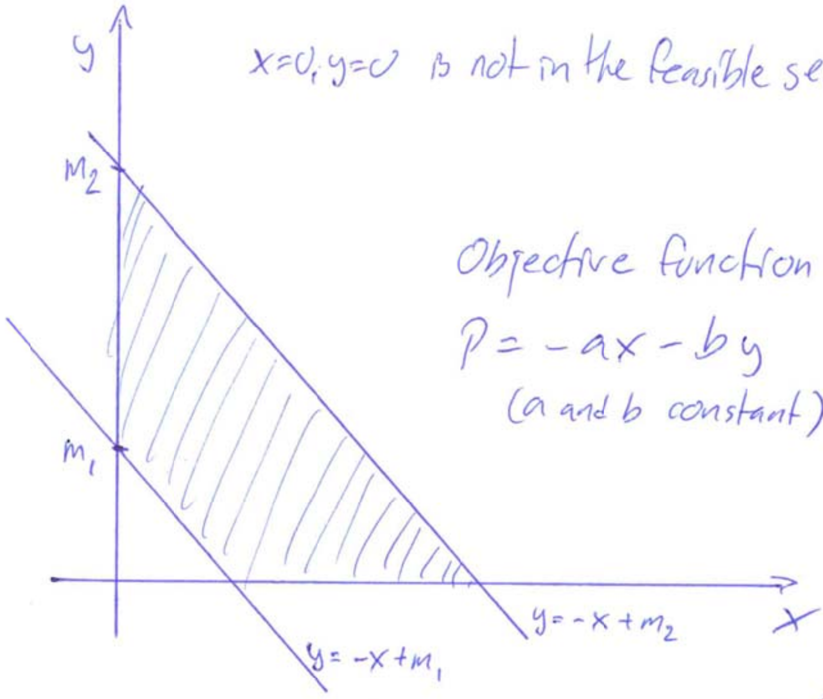
Below are given graphically the constraints for a linear optimization problem. The shaded area is the feasible set, and the lines are given by the respective equations at the bottom. The objective function to minimize is also given in the figure. All constants, *a*, *b*, m_1 and m_2 are positive.



a) Formulate an LP problem in standard form for the given constraints and objective function. (2p)

b) Prepare the model to be solvable by the tableau method. Remember that all variables must always be larger or equal to zero. (3p)

c) Using the given constants, give an upper bound on the optimum of the objective function. (1p)



$x=0, y=0$ is not in the feasible set!

Objective function
$P = -ax - by$
(a and b constant)

$\min P = -ax - by$
s.t. $y \geq -x + m_1$
$y \leq -x + m_2$

with slack and surplus variables

$y + x - s = m_1$
$y + x + t = m_2$

Rewrite

$s = -m_1 + x + y$
$t = m_2 - x - y$

At $x=0, y=0$, $s = -m_1$
Not Good!

Rearrange

$y = m_1 - x + s$
$t = m_2 - x - (m_1 - x + s) = (m_2 - m_1) - s$
$P = -ax - b(m_1 - x + s) = -bm_1 + (b-a)x - bs$

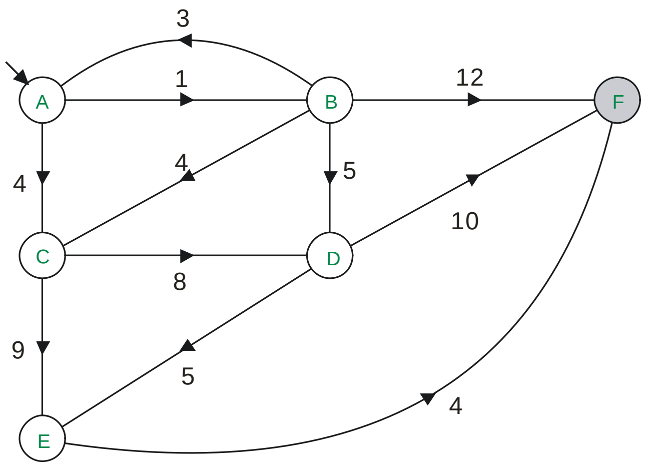
At the origin ($x=0$, $s=0$) y and t are positive. GOOD!
As we see, $P^* \leq -bm_1$

Task 3. Integer Linear Programming Theory

In integer linear programming theory the notion of a *totally unimodular matrix* arises. Describe what this is and why it is important. (3p)

A unimodular matrix *is a square integer matrix the inversion of which is also integer. A* totally unimodular matrix *is a, not necessarily square, matrix for which all non-singular square sub-matrices are unimodular. This definition implies that a totally unimodular matrix only contains -1, 0 and +1 elements. In an integer linear programming problem, if the A-matrix is totally unimodular and the b-vector is integer, then the optimal solution calculated as a linear programming problem will also be integer. Thus, solving the LP-relaxation will immediately solve the original IP-problem.*

Task 4. Discrete Optimization



Above is given a graph, with costs on the edges.

a) Using Dijkstra's algorithm, find the least cost path through the graph. (3p)

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. For b) you need to define a good estimate.

The optimal path is A-B-F.

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[1,A] C[4,A]	A[0,10,-]	B[1,9,A] C[4,10,A]
B[1,A]	C[4,A] F[13,B] D[6,B]	B[1,9,A]	F[13,0,B]: C[4,10,A]: D[6,8,B]
C[4,A]	D[6,B] F[13,B] E[13,C]	F[13,0,B]	D[6,8,B]: C[4,10,A]
D[6,B]	E[11,D] F[13,B]		
E[11,D]	F[13,B]		
F[13,B]			

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 monotonic and not over-estimating, and this example 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).*

Beware the A-B-A-loop. This has to be excluded in some way, else the algorithm may walk around and around forever, but I guess you have already discovered this (as did I).