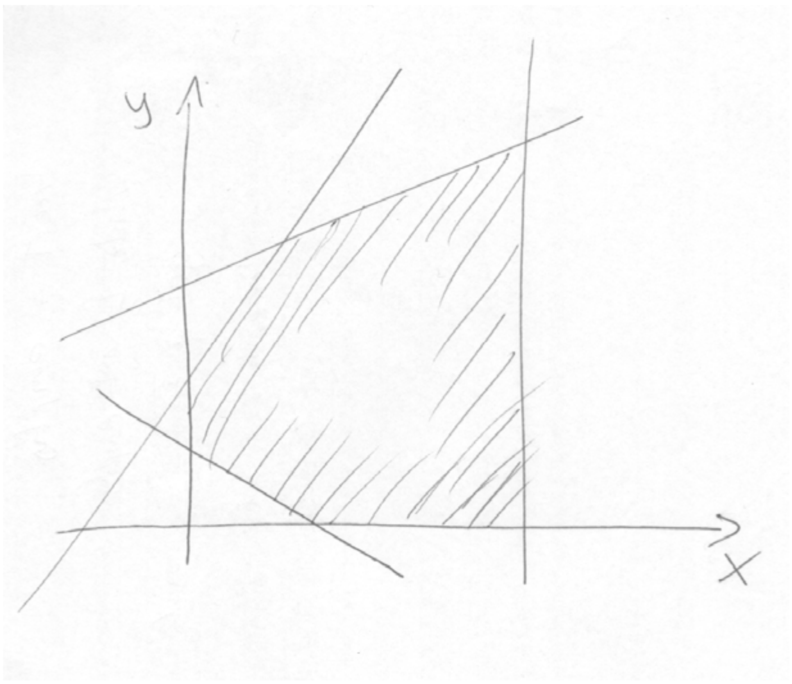
DECO

Discrete Event Control and Optimization

Exam SSY 220, Wednesday May 23, 14:00-18:00, V

Teacher: Bengt Lennartson, (772) 3722

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

Aids: None.

Task 1. Supervisory Control Theory

We pose four requirements on our supervisors. Give those, both formally and informally. Explain what can happen if each is not fulfilled. (4p)

The requirements are:

Within the spec	$L(P\|S) \subseteq L(P\|Sp)$
	$L_m(P\|S) \subseteq L_m(P\|Sp)$
Non-blocking	$L(P\|S) \subseteq \overline{L_m(P\|S)}$
Controllable	$L(P\|S)\Sigma_u \cap L(P) \subseteq L(P\|S)$
Minimally restrictive	$\forall S' \in \mathrm{CNB}(P\|Sp),\ P\|S' \le P\|S$

Here $\mathrm{CNB}(P\|Sp)$ *is the set of all supervisors given the plant* P *and the specification* Sp *that may exist, fulfilling the requirements of controllable, nonblocking and "within the spec".*

The supervisor is there to restrict the plant to stay within the specification, to do it in such a way that some marked state may always be reached (non-blocking) and so that it restricts the plant as little as absolutely necessary (maximally permissive). For us to be able to assess those properties, the supervisor also has to be controllable, *uncontrollable events cannot lead the controlled-system out of the allowed state-space.*

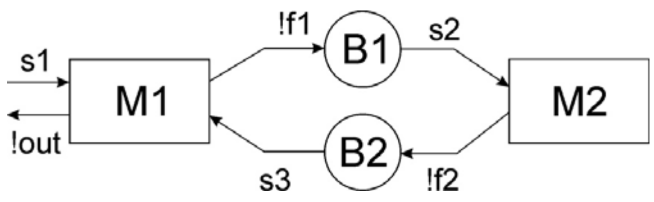
If 1 does not hold, we break the given spec.

If 2 does not hold, we may never reach any marked state.

If 3 does not hold, we may be forced outside the spec due to an uncontrollable event.

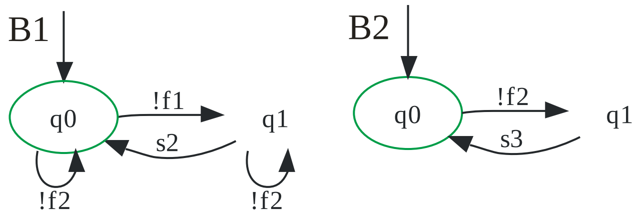
If 4 was not required, the degenerate null supervisor would be a solution.

Task 2. Compositional Supervisor Synthesis



Consider a system consisting of two machines M_1 and M_2 working on parts stored in two buffers B_1 and B_2, see the figure above. Machine, M_1, gets work-pieces (event s_1) from outside the system, processes them, and puts them into B_1 (event $!f_1$). M_I also takes work-pieces from B_2 (event s_3), processes them, and outputs them (event $!out$). Machine M_2 fetches work-pieces (event s_2) from B_1, processes them, and puts them into B_2 (event $!f_2$).

Uncontrollable events are prefixed by an exclamation mark, the other events are controllable. Only the six events described above are present in the system model.



The two buffers can be modeled as above, and these models act as specifications saying that the buffers must never overflow, and we should always be able to empty them.

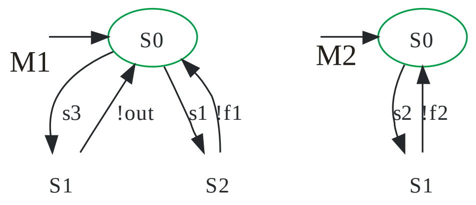
a) Model the two plant components, M_1 and M_2. (2p)

b) Consider the compositional approach, and the *uncontrollable observation equivalence* abstraction, perform the abstraction steps as far as possible. (3p)

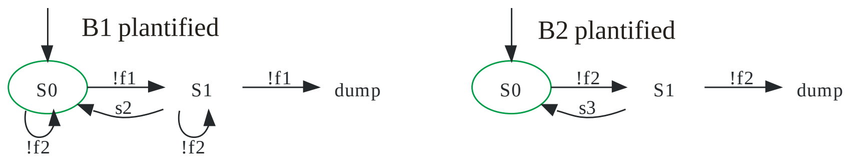
c) Describe, in words, how a supervisor generated by the abstraction-based compositional approach is to be used to control a given system. (1p)

Hint: For abstraction, regard M_1 and focus on local events. Remember that not only individual components can be abstracted, but also compositions of components.

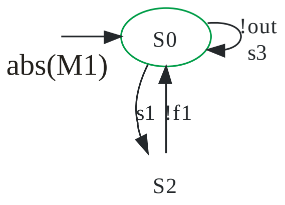
The plant components are most easily modeled as below.

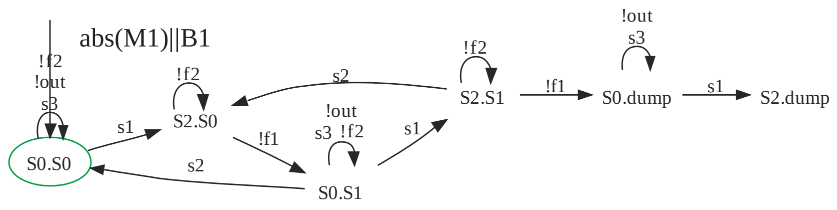


To use the given specifications for compositional synthesis, we first need to turn them into plants; we "plantify". This gives the result below.

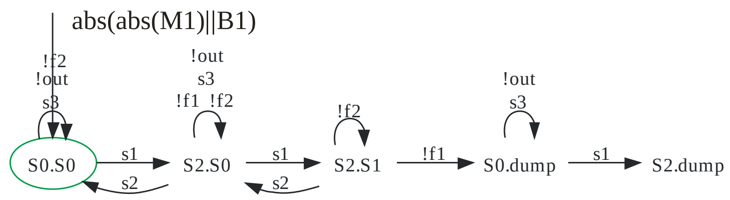


To start abstracting M1, we note that !out is a local uncontrollable event, and so can be abstracted, S0 and S1 of M1 can be merged. This gives us the left automaton below. Now we can note that synchronizing this automaton with the (plantified) B1, makes !f1 a local uncontrollable event, see the right automaton below.

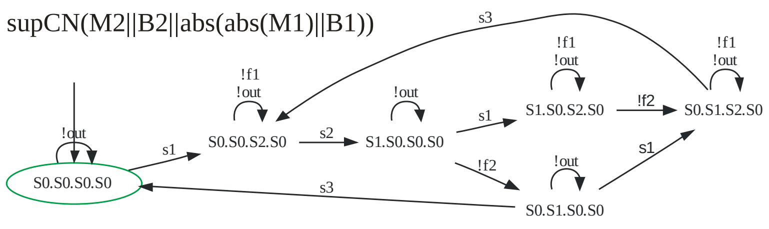




Abstracting !f1 results in the automaton below.



No further abstraction is possible, in particular, !f2 cannot be made a local uncontrollable event, save synchronizing all three remaining automata, which results in an automation with 30 states, too large to handle manually. Nonetheless, the resulting supervisor looks like this:



This supervisor must be used together with the specifications B1 and B2 to when controlling the given plant M1||M2.

Task 3. Linear Programming

The factory where you work is to be upgraded. There are however many different upgrade options and you have proposed using linear programming techniques to decide what to do. As no one else at the factory has any knowledge of linear programming, you have been put in charge of the modeling.

There are two chemical processing machines m_1 and m_2. Either one or both of these should be upgraded. However, the current cooling system can only support one upgraded machine. If both machines are upgraded, the cooling system c must also be upgraded. Also, note that the cooling system may only be upgraded if both machines are upgraded.

The amount of raw material that is processed by the plant is quantified by p [m^3/h]. The amount of finished product is given by f [m^3/h] and is governed by the relation

$$f = kp$$

where k = [0.6, 0.9] depending on the machine upgrade. If both machines are upgraded, k = 0.9, if only one is upgraded k = 0.6.

a) Write down the logical relation between the machines m_1 and m_2, and the cooling system c using logical expressions and then give the equivalent linear constraints using 0-1 variables. (2p)

b) Using the variables in a), model the $f = kp$ using linear expressions.
(hint: use the *big M* method.) (3p)

c) We know that p varies from 0 to 10. How large should the *big M*'s in b) be for the formulation to be "tight"? (2p)

Let m_1, m_2 and c be logical variables that are true whenever the two machines and the cooler, respectively, are upgraded. Then the logical expression is

$$(m_1 \vee m_2) \wedge (m_1 \wedge m_2 \Leftrightarrow c).$$

This requires at least one of m_1 or m_2 to be true, and c is true if and only if both m_1 and m_2 are true. This can be modeled by 0-1 variables δ_1, δ_2 and δ_c that respectively represent the m_1, m_2 and c logical variables. Then the above expression can be modeled as

$$\begin{aligned} \delta_1 + \delta_2 &\geq 1 \\ \delta_c &\geq \delta_1 + \delta_2 - 1 \\ \delta_1 &\geq \delta_c \\ \delta_2 &\geq \delta_c \end{aligned}$$

We are supposed to model that f=0.9p when c is true ($\delta_c = 1$), and f=0.6p when c is false ($\delta_c = 0$). But we cannot model the equality straight off; we need in both cases to model as less-or-equal, and greater-or-equal, respectively. One way to do it is like this:

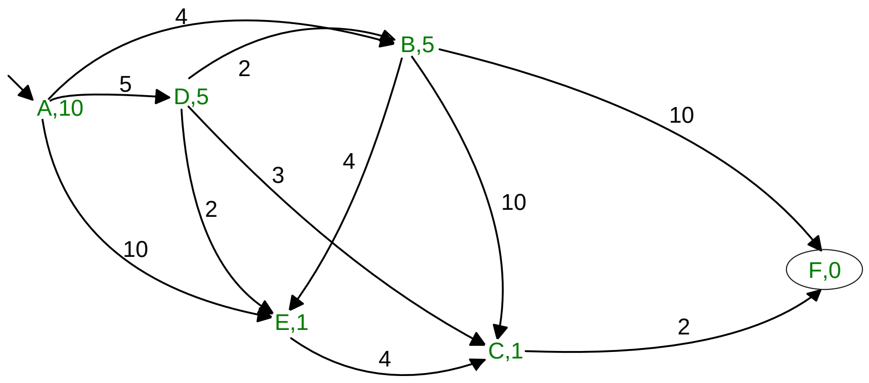
$$\begin{aligned} f &\leq 0.9p + M_1(1-\delta_c) \\ 0.9p - M_2(1-\delta_c) &\leq f \\ f &\leq 0.6p + M_3\delta_c \\ 0.6p - M_4\delta_c &\leq f \end{aligned}$$

To determine tight M's, we need to consider the values f can take as p varies from 0 to 10 and see what constraints this puts on our M's. It suffices to look at the extreme values, 0 and 10. When p=0, irrespective of c, f=0. This tells us that the M's must be larger or equal to 0. When p=10 and $\delta_c = 1$*, then f=9, which tells us that* $3 \le M_3$*, and* $-M_4 \le 3$*. When p=10 and* $\delta_c = 0$*, then f=6, which tells us that* $-3 \le M_1$ *and* $-M_2 \le -3$*. Together, this gives us*

$$M_1 = 0, M_2 = 3, M_3 = 3, M_4 = 0$$

as the tightest M-values we can have.

Task 4. Discrete Optimization



Above is given a graph, with costs 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. (2p)

c) For A* to guarantee to return the optimal path, the heuristic estimate must fulfill a specific property. Describe this property, both formally and intuitively. Does the estimate above fulfill this property? (2p)

d) For A* to guarantee to search as few nodes as necessary, the heuristic estimate must fulfill a specific property. Describe this property, both formally and intuitively. Does the estimate above fulfill this property? (2p)

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

Dijkstra's Algorithm		*A**	
A[0,-]	B[4,A] D[5,A] E[10,A]	A[0,10,-]	B[4,5,A] D[5,5,A] E[10,1,A]:
B[4,A]	D[5,A] C[14,B] E[8,B] F[14,B]	B[4,5,A]	E[8,1,B] F[14,0,B] D[5,5,A] C[14,1,B]
D[5,A]	E[7,D] F[14,B] C[8,D]	E[8,1,B]	D[5,5,A] F[14,0,B] C[12,1,E]
E[7,D]	C[8,D] F[14,B]	D[5,5,A]	E[7,1,D] F[14,0,B] C[8,1,D]
C[8,D]	F[10,C]	C[8,1,D]	F[10,0,C]
F[10,C]		F[10,0,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.

The estimate never overestimates any path from a node to the goal, which this table reveals:

	Estimate	*True min*
A to F	*10*	*10*

B to F	5	10
C to F	1	2
D to F	5	5
E to F	1	6
F to F	0	0

Thus, the estimate is **admissible** and we are guaranteed that the optimal path is found. This we also see since A* returns the same path as does Dijkstras algorithm. Formally, we write $h(n) \le \min\left(h_i^*(n)\right)$, where $h_i^*(n)$ is the true cost from node n along the i:th path.

The following table reveals that the given heuristic is not monotone.

c(x,y)	A, 10	B, 5	C, 1	D, 5	E, 1	F, 0
A, 10		4		5	10	
B, 5			10		4	10
C, 1						2
D, 5		2	3		2	
E, 1			4			
F, 0						

The optimal path is A-D-C-F. 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.

We see above that for instance h(A)=10 but c(A,B)=4 and h(B)=5, which breaks the **monotonicity**, since $h(A) > c(A,B) + h(B)$. This is why A* picks B in the second step, instead of D, as it would have done if the estimate was monotone. Non-monotonicity is also the reason why E is picked before D in the third step. Monotonicity is formally written as $h(n) \le \min\left(c(n,m_i) + h(m_i)\right)$, where $c(n,m_i)$ is the true cost to take the single step from node n to node m_i. Monotonicity guarantees that A* expands the fewest necessary nodes.