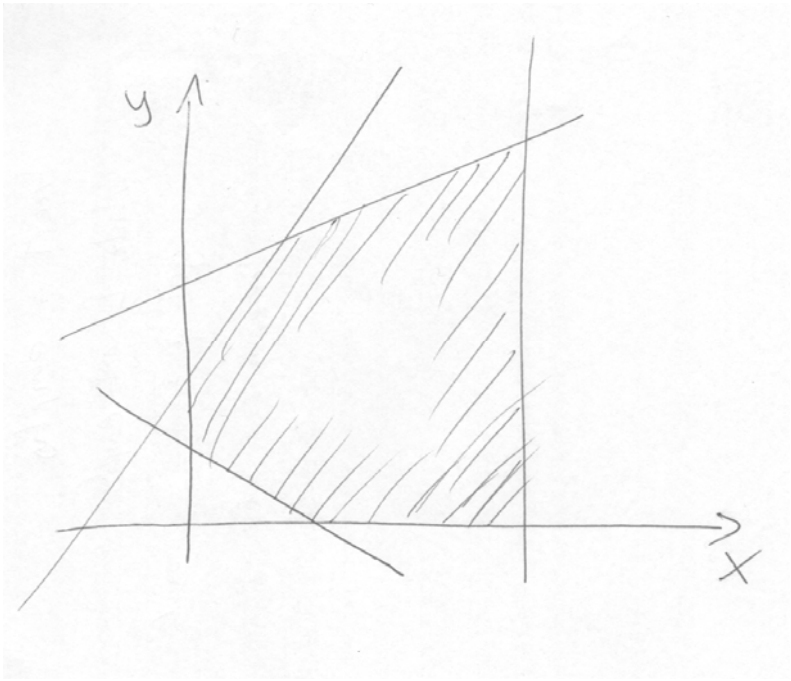
DECO

Discrete Event Control and Optimization

Exam SSY 220, Tuesday May 28, 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.

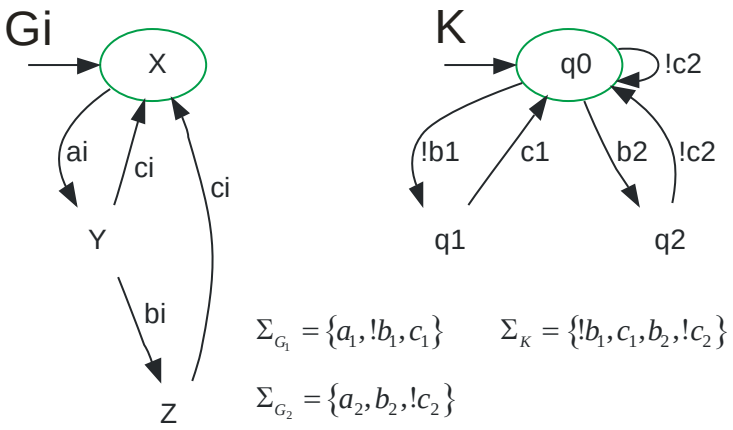
In total the exam comprises 25 credits. For the grades 3, 4 and 5, is respectively required 10, 15 and 20 credits.

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

Aids: None.

Task 1. Controllability

Controllability is an important property in the Supervisory Control Theory; the supervisor must be controllable with respect to the plant and the uncontrollable events. Below is a specification K and a plant component G_i of which there are two (i = 1,2), and the uncontrollable events are !b1, and !c2 (note that the plant components are not identical in this respect). By a clever trick called *plantify* we can remove the distinction between plant and specification while still being able to synthesize a proper supervisor.



$\Sigma_{G_1} = \{a_1, !b_1, c_1\}$ $\Sigma_K = \{!b_1, c_1, b_2, !c_2\}$

$\Sigma_{G_2} = \{a_2, b_2, !c_2\}$

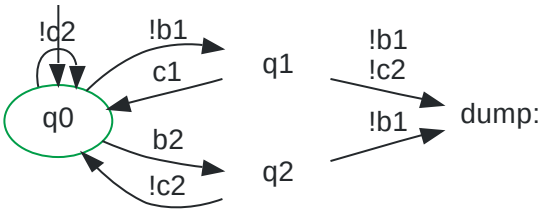
a) Explain controllability and give the formal definition. What can happen if controllability does not hold? (2p)

b) Plantify the given specification K above. (2p)

c) Is K controllable with respect to the G_i (i = 1,2) and the uncontrollable events !b1, and !c2? Explain. (2p)

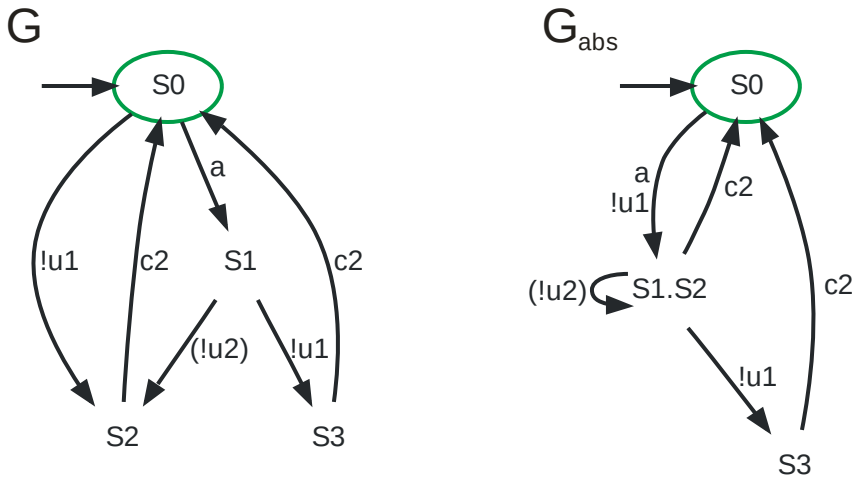
Formally controllability is defined as $L(G\|K)(\Sigma_u \cap \Sigma_G \cap \Sigma_K) \cap L(G) \subseteq L(G\|K)$, *which for equal alphabets is equivalent to* $L(K)\Sigma_u \cap L(G) \subseteq L(K)$. *If controllability does not hold, the closed-loop system may be forced outside the spec due to an uncontrollable event.*

K is not controllable. After a1.!b1.a2, the plant can do !c2 but the specification does not agree. We see this in the plantified version of K below, where !b1.!c2 leads to the dump state.



Task 2. Abstractions for Compositional Verification

Emilia was given the task to calculate a supervisor by compositional synthesis. One of the automata looked like the one to the left below. Emilia happily abstracted away the local uncontrollable event (!u2) into the automaton on the right.



a) Is the abstraction valid for controllability verification? Explain. (3p)

b) Is the abstraction valid for non-blocking verification? Explain. (3p)

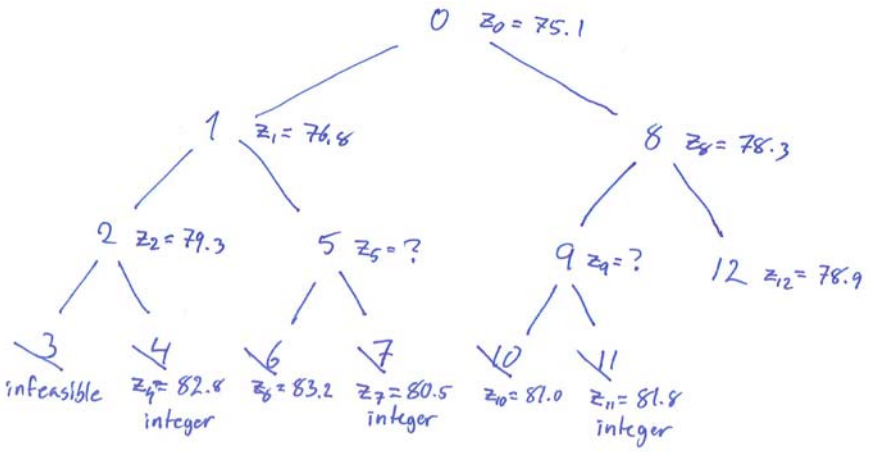
Answering these questions requires finding some test (plant or specification) that does not include the local event (!u), such that it gives different results with G and Gabs. And different results means controllable and uncontrollable, and blocking and non-blocking, respectively. Note that to assess validity for non-blocking verification, the test must fail due to blocking, not due to uncontrollability, and vice versa. So, different tests are needed to assess the two properties.

The abstraction is not valid for controllability verification. We can see this if we consider a test with the language (!u1.c2). This is controllable with G but uncontrollable with Gabs.*

The abstraction is also not valid for non-blocking verification. We can see this if we consider a test which blocks the controllable event a and has the language (!u1.!u1.c2). This is non-blocking with Gabs but blocking with G.*

Task 3. Linear Programming

Emil has an integer programming minimization problem that he tries to solve with LP-relaxations and branch and bound. The search tree Emil has generated so far is shown below. The z_i (with i = 0-12) values are the current LP-relaxation solutions at the respective node; the values for z_5 and z_9 are not disclosed. The nodes are solved in the order of their numbering. The term integer denotes that all variables are integer in the LP-relaxed solution at the respective node. The nodes marked with a slash (\) are cut off and will not be searched further by Emil.



a) Why are nodes 4, 7, and 11 cut off? (1p)

b) Why is node 6 cut off? (1p)

c) Why is node 10 cut off? (1p)

d) Between which bounds lies z_5? (1p)

e) Between which bounds lies z_9? (1p)

f) Between which bounds lies the global solution z^*? (1p)

Nodes 4, 7 and 11 are cut off because they have integer solutions, and continuing searching down those paths will not reveal any integer solutions with better (lower) values. Node 6 is cut off because node 4 already has a better integer solution. Node 10 is cut off since node 7 has an integer solution with better value. $76,8 \le z_5 \le 80,5$ $78,3 \le z_9 \le 80,5$ $78,9 \le z^* \le 80,5$.

Task 4. Discrete Optimization

Emilia works as a project manager. She has a new project to manage, that she has broken down into sub-tasks with some requirements and estimated times. This is summarized in the table below.

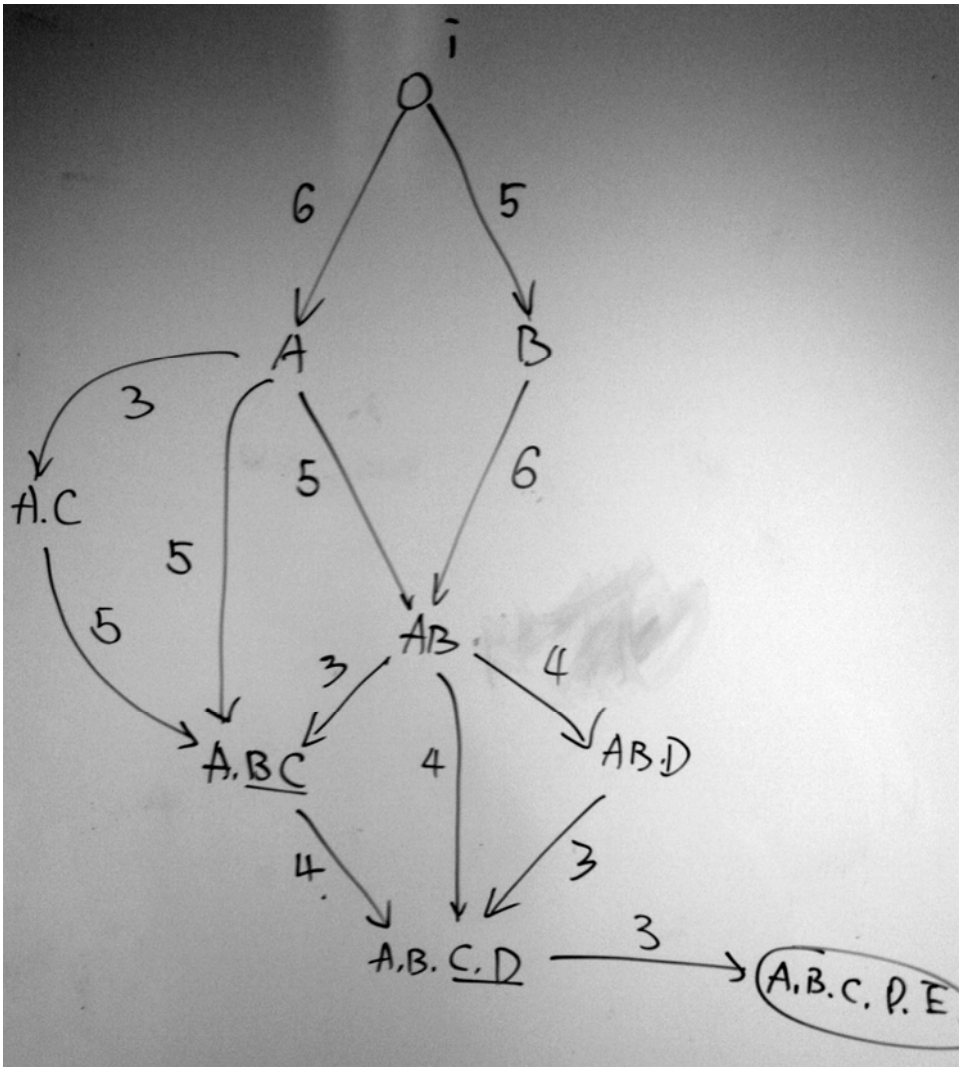
Sub-task	Time	Requirements
A	6	Not simultaneously with B
B	5	Not simultaneously with A
C	3	Can start only after A is done
D	4	Can start only after A and B are done
E	3	Can start only after C and D are done

a) Draw a (precedence) graph that models this project. (3p)

b) Using Dijkstra's algorithm, find the least cost path through the graph. Show on each iteration which node is taken out from and put in to the queue, and also what the queue looks like. (2p)

c) There are more than one equally good least cost paths for this example. Does Dijkstra's algorithm have any preference for one of them? If so, or not so, explain. (2p)

The graph looks something like below. Note that B and C, as well as C and D can be done in parallel, and the time is then the maximum of either, see the edge from A directly to ABC, and from AB to ABCD, respectively..



Above, the node A, for example, represents that the task A is done. Dijkstra's algorithm should perform something like this:

0[0,-]: B[5,0], A[6,0]:

B[5,0]: A[6,0], AB[11,B]:

A[6,0]: AC[9,A], AB[11,B], ABC[11,A]:

AC[9,A]: ABC[11,A], AB[11,B]:

ABC[11,A]: AB[11,B], ABCD[15,ABC]:

AB[11,B]: ABCD[15,ABC], ABD[15,AB]:

ABCD[15,ABC]: ABD[15,AB], ABCDE[18,ABCD]:

ABD[15,AB]: ABCDE[18,ABCD]:

ABCDE[18,ABCD]:

Goal node found --

ABCDE[18,ABCD]: ABCD[15,ABC]: ABC[11,A]: A[6,0]: 0[0,-]:

There are three equally good best paths, 0-A-AB-ABCD-ABCDE, 0-B-AB-ABCD-ABCDE, and 0-A-ABC-ABCD-ABCDE, all with cost 18. The first two reach AB with cost 11, and the third reaches ABC with cost 11. In relation to the first two, Dijkstra's algo, always going for the currently cheapest node, will always choose B in the beginning. Thus, AB is always reached first along 0-B-AB, and the second time AB is encountered, along 0-A-AB, it is not

reached with a better value (it is the exact same value), and so its current parent of AB, that is B, will not be updated. So there is a preference there, the best cost path through AB will never be along the A node.

However, when it comes to 0-B-AB-ABCD-ABCDE versus 0-A-ABC-ABCD-ABCDE, it is only a matter of the order they are taken out from the queue. If AB is taken out before ABC, then the best cost path will go through AB, else it will go through ABC, as above. So there is no preference there.
