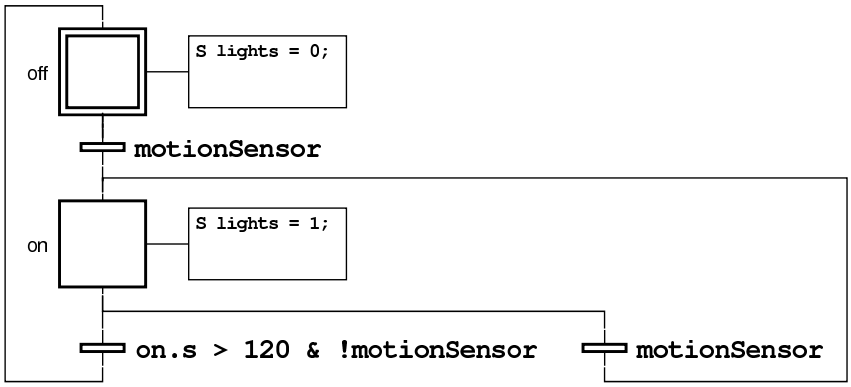

Figure 1

Solutions to the exam in Real-Time Systems 2016-08-23

These solutions are available on WWW: *http://www.control.lth.se/course/FRTN01/*

1.

a. The observer is given by

$$\hat{x}(k+1|k) = \Phi\hat{x}(k|k-1) + \Gamma u(k) + K(y(k) - C\hat{x}(k|k-1))$$

The matrix $\Phi - KC$ is given by

$$\Phi - KC = \begin{pmatrix} 0.5 & 0 \\ -1 & 0 \end{pmatrix} - \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.5 & -k_1 \\ -1 & -k_2 \end{pmatrix}$$

The characteristic equation is given by

$$z^2 + (k_2 - 0.5)z - (k_1 + 0.5k_2) = 0$$

Setting this equal to

$$z^2 = 0$$

gives $k_1 = -0.25$ and $k_2 = 0.5$.

b.
```
y = getY();
setU(u);
x1new = 0.5*x1 - k1*x2 + u + k1*y;
x2new = -x1 - k2*x2 + k2*y;
x1 = x1new;
x2 = x2new;
u = -l1*x1 - l2*x2;
```
Alternatively, you may replace k_1 and k_2 with their numerical values.

2. According to the Grafcet specification the conditions of all transitions connected to the same step must be mutually exclusive. This is not the case for the transitions connected to `on` since `on.s > 120` and `motionSensor` might become true at the same time. A corrected application is shown in Figure 1.

3.

a. Mutual exclusion synchronization is needed when

- multiple threads are accessing shared variables,
- multiple threads are calling non-reentrant code,
- multiple threads are using external units

Synchronization is always needed when there are concurrent accesses to a common resource. Synchronization is not needed if the data is only accessed from one source.

b.

- takes the lock on: this instance of the class
- takes the lock on: this instance of the class. The difference compared to the previous case is that the scope in which the lock is held can be smaller than an entire method
- takes the lock on: the whole class, i.e. the class lock.
- takes the lock on: the object referenced

4. Partial fraction decomposition gives

$$G(s) = e^{-s}\frac{1}{s+1} - \frac{e^{-s}}{2}\frac{2}{s+2}$$

The delay of one time unit corresponds to 2 time samples. Combined with the formula sheet this gives

$$H(z) = \frac{1-e^{-0.5}}{z^2(z-e^{-0.5})} - \frac{1-e^{-1}}{2z^2(z-e^{-1})}$$

5.

a. The control law gives the closed loop system

$$\dot{x}(t) = \begin{pmatrix} -l_1 & 1-l_2 \\ -1 & 0 \end{pmatrix} x(t)$$

The characteristic equation of the closed loop system is

$$s^2 + l_1 s + (1 - l_2) = 0$$

which should be made equal to the desired characteristic equation

$$s^2 + 2s + 1 = 0,$$

i.e., $l_1 = 2$ and $l_2 = 0$.

b. Using the Laplace transfom method gives

$$(sI - A)^{-1} = \begin{pmatrix} s & -1 \\ 1 & s \end{pmatrix}^{-1} = \frac{1}{s^2+1}\begin{pmatrix} s & 1 \\ -1 & s \end{pmatrix}$$

Hence

$$\Phi = e^{Ah} = \mathcal{L}^{-1}(sI - A)^{-1} = \begin{pmatrix} \cos h & \sin h \\ -\sin h & \cos h \end{pmatrix}$$
$$\Gamma = \int_0^h \begin{pmatrix} \cos v \\ -\sin v \end{pmatrix} dv = \begin{pmatrix} \sin h \\ \cos h - 1 \end{pmatrix}$$

With $h = \pi$ this becomes

$$\Phi = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$
$$\Gamma = \begin{pmatrix} 0 \\ -2 \end{pmatrix}$$

c. In order to be able to place the poles arbitrarily the system must be reachable, i.e. the matrix $W_c = [\Gamma \ \Phi\Gamma]$ must have full rank. In our case

$$W_c = \begin{pmatrix} 0 & 0 \\ -2 & 2 \end{pmatrix}$$

which has the determinant equal to 0. It is easily seen that the discrete-time system has both its poles in -1. Only one of these poles is effected by the control signal. The original continuous-time systems is, however, controllable. This can be seen from continuous-time controllability matrix $[B \ AB]$ which has full rank. Hence, through ZOH-sampling with this particular sampling period the controllability is lost.

6. Factorizing the denominators gives the matching between the pulse-transfer functions and the pole diagrams:

$$H_1(z) = \frac{1}{z^2 + 0.8z + 0.07} = \frac{1}{(z+0.1)(z+0.7)}$$
$$H_2(z) = \frac{1}{z^2 - 0.8z + 0.07} = \frac{1}{(z-0.1)(z-0.7)}$$
$$H_3(z) = \frac{1}{z^2 - 0.2z + 0.01} = \frac{1}{(z-0.1)^2}$$
$$H_4(z) = \frac{1}{z-1}$$
$$H_5(z) = \frac{1}{z^2 - 0.8z + 0.52} = \frac{1}{(z-0.4+0.6i)(z-0.4-0.6i)}$$
$$H_6(z) = \frac{1}{z-1.2}$$

Only A and B are oscillatory, meaning that they have complex poles or poles on the negative real axis. B oscillates at half the sampling frequency, which means that it has poles on the negative real axis and we have f-B and d-A.

a has a pole outside the unit circle, resulting in the unstable system E.

e has a pole in 1, resulting in the integrator C.

b, c, D and F remain. The slowest pole (farthest from the origin) is slower in c than in b. Hence, c is paired with the slower step response D and b is paired with F.

The correct answer is:

1-f-B

2-c-D

3-b-F

4-e-C

5-d-A

6-a-E

7. Using the formula

$$f_{\text{fundamental}} = |(f + f_s/2) \bmod (f_s) - f_s/2|$$

with $f_{\text{fundamental}} = 0.1, f = 100$ and $f_s \approx 50$ we obtain $f_s = (100 - 0.1)/2 = 49.95$ and $f_s = (100 + 0.1)/2 = 50.05$.

8 a. Each element, a, in the matrices is rounded in the following way:

$a_{approx} = \frac{\text{round}(a \cdot 2^3)}{2^3}$

The approximated system is

$$\begin{aligned} x(k+1) &= \begin{pmatrix} 0.875 & 0.875 \\ 0.125 & 0.500 \end{pmatrix} x(k) + \begin{pmatrix} 3.375 \\ 2.125 \end{pmatrix} u(k) \\ y(k) &= \begin{pmatrix} 5.125 & 7.375 \end{pmatrix} x(k) \end{aligned}$$

8 b. The poles of the systems are given by $\det(zI - A) = 0$.

For the original system we have

$|zI - A| = (z - 0.82)(z - 0.50) - 0.07 \cdot 0.86 = z^2 - 1.32z + 0.41 - 0.07 \cdot 0.86 = 0 \Rightarrow z_1 = 0.953, z_2 = 0.367$

For the approximated system we get

$|zI - A| = (z - \frac{7}{8})(z - \frac{4}{8}) - \frac{7}{64} = z^2 - \frac{11}{8} + \frac{21}{64} = 0 \Rightarrow z_1 = 1.068, z_2 = 0.307$

The approximation causes one of the poles to move outside of the unit circle, meaning that the approximated system is unstable, unlike the original system which is stable.

9.

a. Under EDF the task set is schedulable if

$$U = \sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq 1$$

In this case

$$U = 1/3 + 2/8 + 5/20 = 0.833$$

Hence, the task set is schedulable under EDF.

b. Using the original Lui and Layland sufficient-only test we get that

$$U = 0.833 > 3(2^{1/3} - 1) = 0.7798$$

Hence, using this test we cannot tell if the task set is schedulable or not.

Using the hyperbolical sufficient-only test we get

$$\prod_{i=1}^{i=n}(\frac{C_i}{T_i} + 1) = (0.333 + 1)(0.25 + 1)^2 = 2.083 > 2$$

Hence, also using this test we cannot tell if the task set is schedulable or not.

Using the response time analysis starting with the highest priority task we get

$$R_A^0 = 0$$

$$R_A^1 = C_A = 1 < 3$$

$$R_B^0 = 0$$

$$R_B^1 = C_B = 2$$

$$R_B^2 = 2 + \left\lceil \frac{2}{3} \right\rceil 1 = 2 + 1 = 3$$

$$R_B^3 = 2 + \left\lceil \frac{3}{3} \right\rceil 1 = 2 + 1 = 3 < 8$$

$$R_C^0 = 0$$

$$R_C^1 = C_C = 5$$

$$R_B^2 = 5 + \left\lceil \frac{5}{3} \right\rceil 1 + \left\lceil \frac{5}{8} \right\rceil 2 = 5 + 2 + 2 = 9$$

$$R_B^3 = 5 + \left\lceil \frac{9}{3} \right\rceil 1 + \left\lceil \frac{9}{8} \right\rceil 2 = 5 + 3 + 4 = 12$$

$$R_B^4 = 5 + \left\lceil \frac{12}{3} \right\rceil 1 + \left\lceil \frac{12}{8} \right\rceil 2 = 5 + 4 + 4 = 13$$

$$R_B^4 = 5 + \left\lceil \frac{13}{3} \right\rceil 1 + \left\lceil \frac{13}{8} \right\rceil 2 = 5 + 5 + 4 = 14$$

$$R_B^5 = 5 + \left\lceil \frac{14}{3} \right\rceil 1 + \left\lceil \frac{14}{8} \right\rceil 2 = 5 + 5 + 4 = 14 < 20$$

Hence, the task set is schedulable under fixed-priority rate-monotonic scheduling.

c. Let's begin with the Lui and Layland test. With T_B as unknown we get the equation

$$0.333 + 2/T_B + 0.25 = 3(2^{1/3} - 1) = 0.7798$$

This has the solution $T_B = 10.16$. This is longer than the original value 8, which we already know is OK, so this does not give anything.

If we do the same thing with the hyperbolic test we get

$$1.333 * 1.25 * (1 + 2/T_B) = 2$$

This has the solution $T_B = 9.985$. Also this is longer than the original value, which we already know is OK, so again this does not give anything.

If we now use the upper bound on the response time and choose T_B such that the bounds for Task B and Task C equal their corresponding deadlines we get the following two equations

$$\bar{R}_B = \frac{2 + 1 * (1 - 0.333)}{1 - 0.333} = T_B$$

This has the solution $T_B = 3.998$ which is smaller than 8. However, we must now also ensure that the deadline for Task C is larger than the upper bound. Hence, we get

$$\bar{R}_C = \frac{5 + 1 * (1 - 0.333) + 2(1 - 2/T_B)}{1 - (0.333 + 2/T_B)} = 20$$

This has the solution $T_B = 6.346$. Hence, we know for sure that $T_B = 6.346$ is a feasible candidate as the shortest period for task B. If one spends the time calculating the shortest period using response-time analysis the true shortest period is 5.

10.

a. The support for condition synchronization in normal synchronized methods in Java can not distinguish between different conditions. This means in this case, e.g., that when a Producer enters a negative integer, also the ConsumerB processes will be woken up, which only causes them to wait again. A similar situation holds for ConsumerA processes for values that are positive. Also, when a consumer gets a value it will not only wake up the producers, but also other consumers.

b.
```
public class Buffer {
  private Semaphore mutex;
  private ConditionVariable nonFull, fullForA, fullForB;
  private int data;
  private boolean full = false;


  public Buffer() {
    mutex = new Semaphore(1);
    nonFull = new ConditionVariable(mutex);
    fullForA = new ConditionVariable(mutex);
    fullForB = new ConditionVariable(mutex);
  }

  public void put(int inData) {
    mutex.take();
    while (full) {
      try {
```

```
        nonFull.cvWait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    data = inData;
    full = true;
    if (data < 0) {
      fullForA.cvNotifyAll();
    } else {
      fullForB.cvNotifyAll();
    }
    mutex.give();
  }

  public int getNegative() {
    mutex.give();
    while (!full) {
      try {
        fullForA.cvWait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    full = false;
    nonFull.cvNotifyAll();
    mutex.give();
    return data;
    }

  public int getPositive() {
    mutex.give();
    while (!full) {
      try {
        fullForB.cvWait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    full = false;
    nonFull.cvNotifyAll();
    mutex.give();
    return data;
    }
}
```

The other classes remain the same. In this solution it is OK to replace all occurrences of `cvNotifyAll()` with `cvNotify()` since it is always known that the single thread that is woken up is actually waiting for the condition that has occurred.