Prof. Bernd Finkbeiner
Dr. Calogero Zarba
Moritz Hahn

Sample Solution for the Sample Final Questions

**Embedded Systems**

# About this Document

This document gives sample solutions for the sample questions for the final examn Embedded Systems, SS 07.
For each problem we will give a short version and/or a notice what you would have to write here for similar exercises in the final examn. Additionally if neccessary, we will give some explanations about the solutions, which you would not have to write in the examn – and probably should not, because you won't have the time anyway.

## Problem 1 (Statechart)

### Solution

Draw figurs 1 and 2.

### Explanation

A very basic version of the pedometer is given in figure 1. The upper part of the model is responsible for updating the value of the `distance`: Each time, `signal` becomes true, that is, the user has made a step, the `step_length` is added, and control changes from `wait_for_signal_on` to `wait_for_signal_off`, where we wait for the `signal` to go off again. After it is false, we go back to `wait_for_signal_on` and are ready for the next step.
In the lower parallel part of the Statechart, `time` and `speed` are updated. As soon as a `tick` event occurs, we know that 0.1 seconds have passed, so we add this value to `time` while going from `update_time_and_speed` to `A`. Immediately this state is left again, the `speed` is updated and we are back in `update_time_and_speed` again. Notice that the `speed` is the average speed since the Statechart was started, that is since the batteries where put in, so to speak. Also notice, that the semantic used corresponds to the Statechart semantics and *not* to the Simulink one.
A Statechart containing the advanced features is given in figure 2: Here, a boolean variable `on_off_switch` is added,

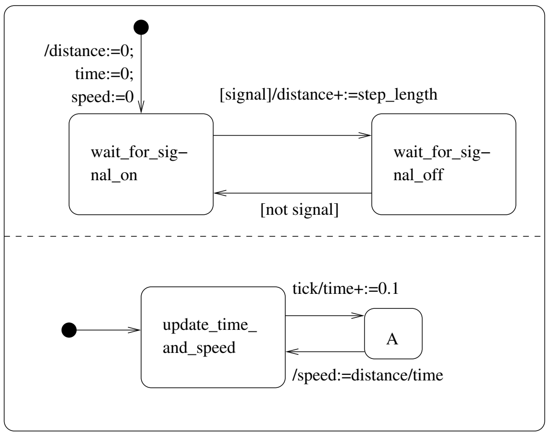


Figure 1: Problem 1: Statechart for basic pedometer

which turns on the pedometer when it is true and immediately turns it off when it becomes false. Each time the pedometer is switched on, the values `distance`, `time` and `speed` are reset. Additionally, a `pause` button has been implemented: If this button is false, the pedometer will work normally. However, if it is true, the time will be stopped and no steps be counted till the `pause` switch is false again.
In the final examn we will tell you exactly what features to implement, of course.

## Problem 2 (Kahn process networks)

*Notice: There was an error in the problem, we meant* $\frac{n(n+1)}{2}$ *instead of* $\frac{n(n-1)}{2}$.

## Solution

In the examn, you would have to write down the programs of the section “Process definitions” and draw figure 6. However, if you are not completely shure that your answere will be 100 percent correct, you could add a short explanation to reduce the ammount of points we will reduce if it is not.

## Explanation

### Kahn process networks in general

A Kahn process network (KPN) is a model describing a network of processes. It is a graph, where nodes correspond to processes whereas the connections – channels – between these processes correspond to edges between them. The channels are one-directional, so, we have a directed graph. A node of a KPN should be labelled with the program that runs on it. This program can be described similar a usual C-program; in general, a single node can do any calculation. Communication with other processes is done by sending on channels or receiving from channels. If a process sends on a channel, the message is placed on the channel immediately, in a FIFO way. If a process tries to receive from a channel and there are messages, it receives the oldest one. However, if there is no message on the channel yet, the process has to wait until there is some, therefore blocking the process node till a message arrives. It is not possible to wait on several input channels at the same time or to check the status of a channel without waiting on it. Because of these restrictions, the calculation of a KPN is deterministic (given the calculations of the processes are).

### The basic processes

In this exercise, one part is to define basic processes that a) add two numbers, b) multiply two numbers, c) duplicate a number d) generate a constant, put it out and then just forward its input e) just receive numbers. In the following, possible programs for these processes are given:
a) “+”: Adding two numbers. We assume that we have two input channels `in1`, `in2` and one output channel `out`

```
for (;;) {
  a := wait(in1);
  b := wait(in2);
  send(a + b, out);
}
```

The process waits for the first number to occur on the first input channel (or just takes it, if it is already there) and then for the second on the second input channel. Then it adds them up and sends them to the output channel. We do this infinitely often.
b) “*”: Multiplying two numbers. Same assumptions as before, almost the same, except the operator.

```
for (;;) {
  a := wait(in1);
  b := wait(in2);
  send(a * b, out);
}
```

c) “D”: Duplicating a number. We assume that we have one input channel `in` and two output channels `out1`, `out2`.

```
for (;;) {
  a := wait(in);
  send(a, out1);
  send(a, out2);
}
```

We wait for an input and then just send it unmodified to both the output channels. We do this all the time.
d) “Ci”: Constant. We assume we have one input channel `in` and output channel `out`. Also, i shall be the constant to be send initially.

```
send(i, out);
for (;;) {
  a := wait(in);
  send(a, out);
}
```

Initially, we just send the constant i. Then, infinitely often we wait for a number and just forward it to the output channel.
e) “S”: Sink process: We assume we have one input channel `in`.

```
for (;;) {
  wait(in);
}
```

We just wait for numbers infinitely often and throw them away as they occur. Remember that we asked you to restrict yourself to this set of processes. If you use more powerful processes which we did not allow in the exercise, you will inevitably loose a lot of points.
An example for Kahn petri nets that demonstrates how these processes work and interact is given in figure 3.

### General scheme for recursive functions

Recursive functions that are defined as
$f(0) = C0$
$f(1) = C1$
$\dots$
$f(m-1) = Cm-1$
$f(n) = \dots$ for $n \geq m$
where $f(n)$ depends on $f(n-1), \dots, f(n-m)$ can be calculated iteratively in the following way: First, one initializes an array of size $m$ with the values of $f(m-1), \dots, f(0)$ (from left to right). Then, one can calculate the value of $f(m)$ from the values contained in that array. If one has done this, one can shift the whole array to the right, thereby shifting out $f(0)$ and then one can place $f(m)$ as the leftmost value. After this, $f(m+1)$ can be calculated from the values that are in the array now, and so on. Saving all values that are dropped out of the array, one gets $f(0), f(1), \dots$ till one stops the calculation.
This method is modelled in the KPN given in figure 4. The “Recursion Box” models the array described above. The “Calculation Box” calculates the new values $f(n)$, for $n \geq m$. Finally, the values are sent to sink process $S$.
Instantiating the “Recursion Box” for a concrete function should be straight forward. In the “Calculation Box” one has to model the mathematical operations used in the definition of the function. That is, if the function is for example $f(n) = f(n-1) + f(n-2)$, one would just connect the $(f(n))$ edge as the output of a “+” process which gets the $(f(n-1))$ and $(f(n-2))$ edges as its inputs. More complex functions can be modelled in a similar way. Sometimes you will have to refere to the value $n$ that is, the current value of the function parameter. This can be done as described in figure 5. This Kahn network will initially put out $m$, because this is the value at which the recursion starts. Then, it will increment this value and so put out $m+1$, and so on.

### Building the Kahn process network for the exercise

The original function definition was $f(n) = \frac{n(n+1)}{2}$. What we actually wanted you to do is to transform this function to a recursive definition and then transform this to a KPN.
It is $f(n) = \frac{n(n+1)}{2} = \sum_{k=1}^{n} k$. This sum can be transformed into the recursion
$f(0) = 0$
$f(n) = n + f(n-1)$, $n \geq 1$
So, we have $m = 1$ and can build a KPN from it as described before. It is given in figure 6.

## Problem 3 (Markov processes)

### Solution and Explanation

As seen in the lecture, one has to solve the equation set

$$s_{\lim}(A) = s_{\lim}(A) \cdot t(A,A) + s_{\lim}(B) \cdot t(B,A) \wedge s_{\lim}(B) = s_{\lim}(A) \cdot t(A,B) + s_{\lim}(B) \cdot t(B,B) \wedge s_{\lim}(A) + s_{\lim}(B) = 1$$

where $t(A,A) = 0.5, t(A,B) = 0.5), t(B,A) = 0.6, t(B,B) = 0.4$. The solution is $s_{\lim}(A) = \frac{6}{11}, s_{\lim}(B) = \frac{5}{11}$.

## Problem 4 (Scheduling)

### Solution

- Consider task set $\{\tau_1, \tau_2\}$ with $T_1 = 10, C_1 = 5, T_2 = 15, C_2 = 6$
- using RM scheduler, priority of $\tau_1$ should be larger than that of $\tau_2$
- so, RM leads to a miss of the deadline of $\tau_2$, as seen here:
- (now draw figure 7)
- however, because $U = \frac{5}{10} + \frac{6}{15} = 0.9 \leq 1$ the task set can be scheduled by EDF (Earliest Deadline First), as was proven in lecture

### Explanation

Schedulability by RM scheduling is only guaranteed if $U \leq n \cdot (2^{\frac{1}{n}} - 1)$ where $n$ is the number of processes. However, EDF scheduling algorithms can schedule task sets with $U \leq 1$. So, to prove the non-optimality of RM, one only needs to give a task set with $n \cdot (2^{\frac{1}{n}} - 1) \leq U \leq 1$ that can't be scheduled by RM schedulers. Such a task set is for example $\{\tau_1, \tau_2\}$ with $T_1 = 10, C_1 = 5, T_2 = 15, C_2 = 6$. Trying to apply an RM scheduler leads to a miss of the deadline, as seen in figure 7. However, because $U = \frac{5}{10} + \frac{6}{15} = 0.9 \leq 1$, it it schedulable by EDF algorithms.

## Problem 5 (Büchi Automata)

### Solution

Draw figure 8.

### Explanation

The automaton is given in figure 8. It works the following way: The prefix $(ab)^*$ is read in the two states 1 and 2. Finally, $c$ is read such that the automaton changes to state 3 where an infinity sequence of $c$ is read. Notice that 3 is the only accepting states, so finally a change to this state must happen for the word to be accepted. Also, once it is entered it will be entered infinitely often if infinitely many $c$ are read.
However, such an exercise won't appear in the final, because it belongs to point 16, which we excluded from the examn.

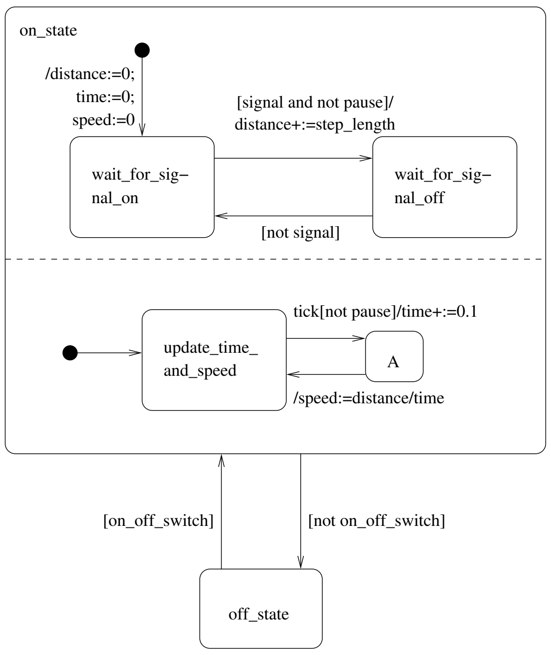
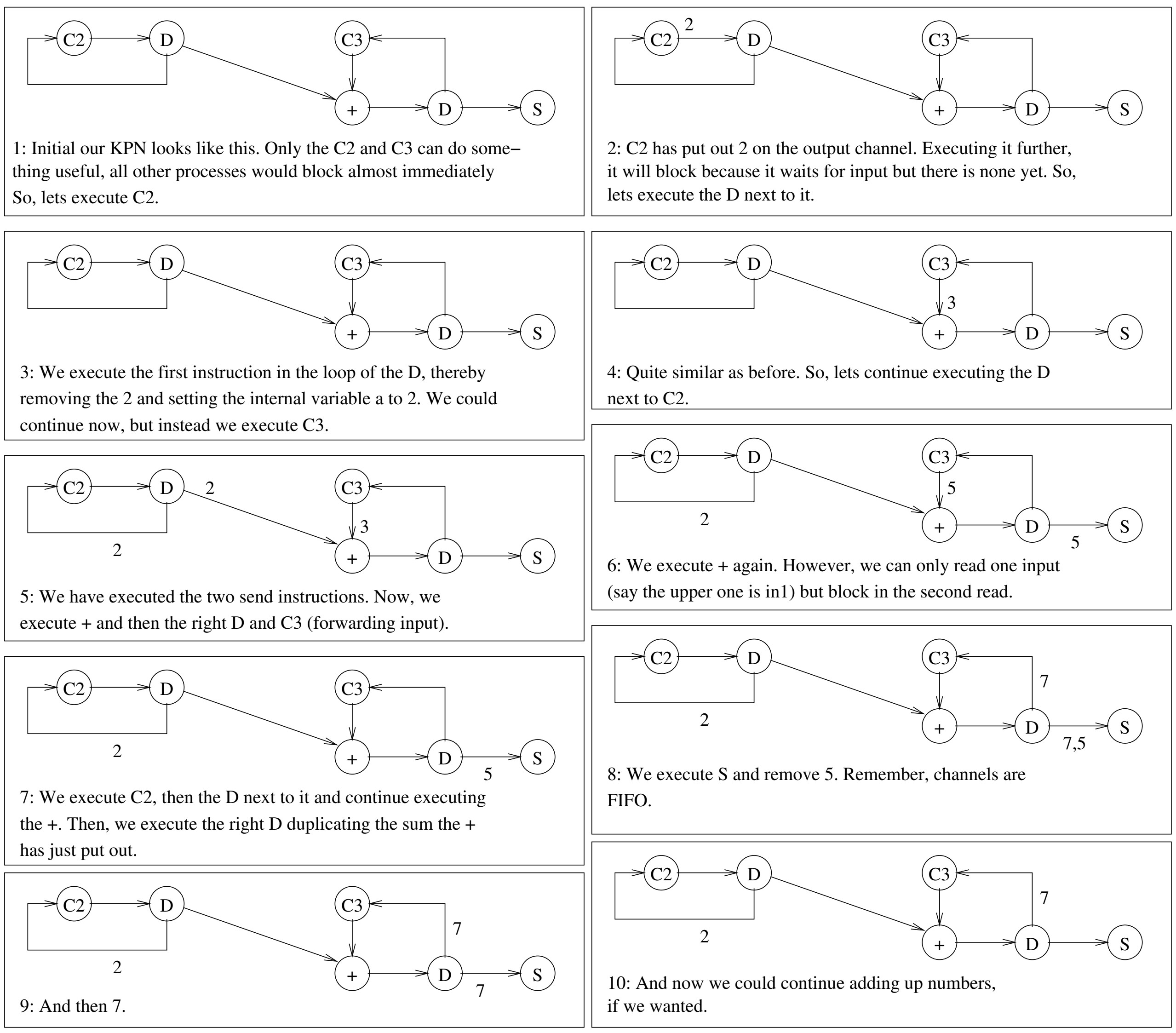


Figure 2: Problem 1: Statechart for pedometer with additional features



1: Initial our KPN looks like this. Only the C2 and C3 can do something useful, all other processes would block almost immediately So, lets execute C2.

2: C2 has put out 2 on the output channel. Executing it further, it will block because it waits for input but there is none yet. So, lets execute the D next to it.

3: We execute the first instruction in the loop of the D, thereby removing the 2 and setting the internal variable a to 2. We could continue now, but instead we execute C3.

4: Quite similar as before. So, lets continue executing the D next to C2.

5: We have executed the two send instructions. Now, we execute + and then the right D and C3 (forwarding input).

6: We execute + again. However, we can only read one input (say the upper one is in1) but block in the second read.

7: We execute C2, then the D next to it and continue executing the +. Then, we execute the right D duplicating the sum the + has just put out.

8: We execute S and remove 5. Remember, channels are FIFO.

9: And then 7.

10: And now we could continue adding up numbers, if we wanted.

Figure 3: Problem 2: Playing around with the basic processes

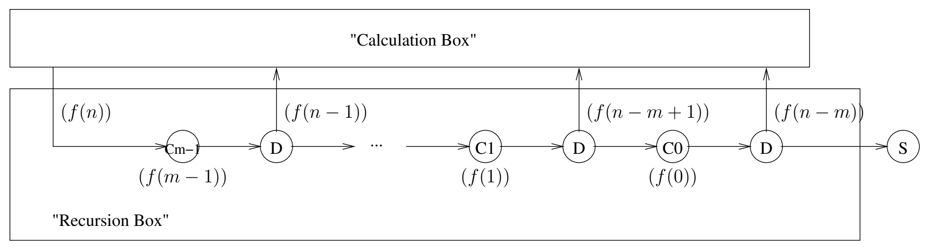


Figure 4: Problem 2: Modelling recursive functions with Kahn process networks

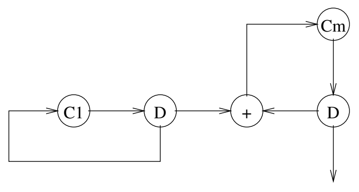


Figure 5: Problem 2: Calculating $n$

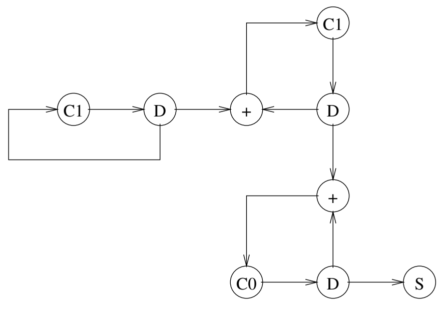


Figure 6: Problem 2: Kahn process network for $\frac{n(n-1)}{2}$

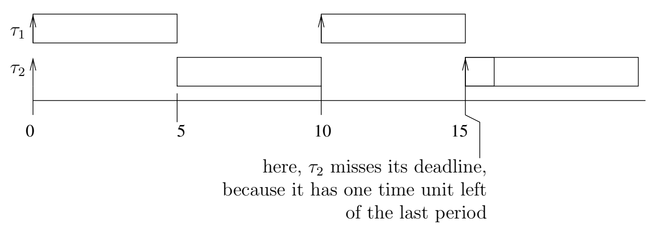


Figure 7: Problem 4: RM is not optimal

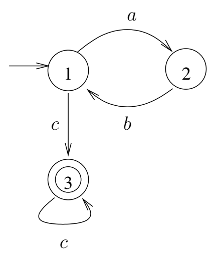


Figure 8: Problem 5: Büchi automaton for $(ab)^*c^\omega$