p73, ex5

5. Consider the deterministic finite-state machine in Figure 3.14 that models a simple traffic light.

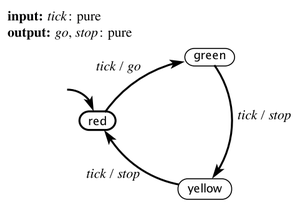


Figure 3.14: Deterministic finite-state machine for Exercise 5

(a) Formally write down the description of this FSM as a 5-tuple:

$$(States, Inputs, Outputs, update, initialState).$$

(b) Give an execution trace of this FSM of length 4 assuming the input *tick* is *present* on each reaction.

(c) Now consider merging the red and yellow states into a single stop state. Transitions that pointed into or out of those states are now directed into or out of the new stop state. Other transitions and the inputs and outputs stay the same. The new stop state is the new initial state. Is the resulting state machine deterministic? Why or why not? If it is deterministic, give a prefix of the trace of length 4. If it is nondeterministic, draw the computation tree up to depth 4.

Solution: The FSM description is:

$$\begin{aligned} States &= \{\text{red}, \text{yellow}, \text{green}\} \\ Inputs &= (\{tick\} \to \{present, absent\}) \\ Outputs &= (\{go, stop\} \to \{present, absent\}) \\ initialState &= \text{red} \end{aligned}$$

The update function is defined as:

$$update(s,i) = \begin{cases} (\text{green}, go) & \text{if } s = \text{red} \wedge i(tick) = present \\ (\text{yellow}, stop) & \text{if } s = \text{green} \wedge i(tick) = present \\ (\text{red}, stop) & \text{if } s = \text{yellow} \wedge i(tick) = present \\ (s, absent) & \text{otherwise} \end{cases}$$

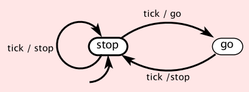
(b) Give an execution trace of this FSM of length 4 assuming the input *tick* is *present* on each reaction.

Solution:

$$\text{red} \xrightarrow{tick/go} \text{green} \xrightarrow{tick/stop} \text{yellow} \xrightarrow{tick/stop} \text{red} \xrightarrow{tick/go} \cdots$$

(c) Now consider merging the red and yellow states into a single stop state. Transitions that pointed into or out of those states are now directed into or out of the new stop state. Other transitions and the inputs and outputs stay the same. The new stop state is the new initial state. Is the resulting state machine deterministic? Why or why not? If it is deterministic, give a prefix of the trace of length 4. If it is non-deterministic, draw the computation tree up to depth 4.

Solution: The resulting state machine is given below. It is non-deterministic because there are two distinct transitions possible from state stop on input *tick*.



We have renamed the green state go. The computation tree for this FSM, up to depth 4, is given below:

L&S 351 ex 3

3. This problem compares RM and EDF schedules. Consider two tasks with periods $p_1 = 2$ and $p_2 = 3$ and execution times $e_1 = e_2 = 1$. Assume that the deadline for each execution is the end of the period.

(a) Give the RM schedule for this task set and find the processor utilization. How does this utilization compare to the Liu and Leland utilization bound of (11.2)?

Solution: The RM schedule is shown below:

The utilization is given by

$$U = 1 - 1/6 \approx 833\%$$

The utilization bound if $n = 2$ is $n(2^{1/n} - 1) \approx 0.828$. Thus, utilization is larger than the utilization bound, so we have no assurance that the RM schedule is feasible.

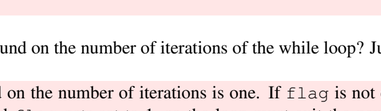
(b) Show that any increase in e_1 or e_2 makes the RM schedule infeasible. If you hold $e_1 = e_2 = 1$ and $p_2 = 3$ constant, is it possible to reduce p_1 below 2 and still get a feasible schedule? By how much? If you hold $e_1 = e_2 = 1$ and $p_1 = 2$ constant, is it possible to reduce p_2 below 3 and still get a feasible schedule? By how much?

Solution: In the first three time units, the RM schedule must execute task 1 twice, because under the RM principle, it has highest priority and it has become enabled twice in this time period. With $e_1 = 1$, this leaves exactly one time unit to execute task 2 in its first period. Thus, any increase in e_2 will result in task 2 missing its deadline at time 3. Any increase in e_1 will leave less than one time unit for task 2 in its first period, resulting again in a missed deadline.

Holding e_1, e_2, and p_2 constant, we can reduce p_1 to 1.5 and still get a feasible schedule. Holding e_1, e_2, and p_1 constant, we can reduce p_2 to 2 and still get a feasible schedule. In both cases, no further reduction is possible because at this point we have 100% utilization.

(c) Increase the execution time of task 2 to be $e_2 = 1.5$, and give an EDF schedule. Is it feasible? What is the processor utilization?

Solution: The EDF schedule is:



The schedule is feasible and the utilization is 100%.

L&S p455, ex2

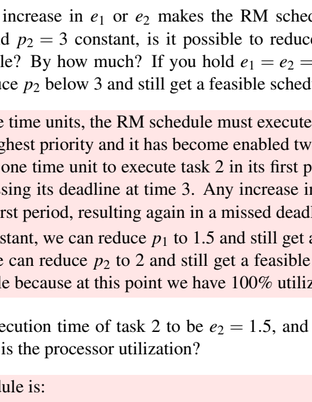
2. Consider the program given below:

```
1  void testFn(int *x, int flag) {
2    while (flag != 1) {
3      flag = 1;
4      *x = flag;
5    }
6    if (*x > 0)
7      *x += 2;
8  }
```

In answering the questions below, assume that x is not NULL.

(a) Draw the control-flow graph of this program. Identify the basic blocks with unique IDs starting with 1.

Solution:



(b) Is there a bound on the number of iterations of the while loop? Justify your answer.

Solution: The bound on the number of iterations is one. If flag is not equal to 1 initially, the loop gets executed and flag gets set to 1, so the loop must exit the next time the condition is evaluated.

(c) How many total paths does this program have? How many of them are feasible, and why?

Solution: This program has a total of 4 paths, corresponding to 2 choices of the conditional in the while loop and 2 choices for the conditional in the if-statement. 3 of these paths are feasible — the only infeasible path is the path 1-2-1-3-5 corresponding to executing one iteration of the while loop (in which *x is set to 1) and the else branch of the conditional *x > 0. This cannot be executed since *x is greater than 0 after executing one iteration of the loop.

(d) Write down the system of flow constraints, including any logical flow constraints, for the control-flow graph of this program.

Solution: The system of flow constraints is as follows: (the logical flow constraint is given later)

$$\begin{aligned} x_1 &= 2 \\ x_1 &= d_{12} + d_{13} \\ x_2 &= 1 \\ x_2 &= d_{12} = d_{21} \\ x_3 &= d_{13} = d_{34} + d_{35} \\ x_4 &= d_{34} = d_{45} \\ x_5 &= d_{35} + d_{45} \end{aligned}$$

Even though this program has a loop, since the loop body is executed at most once, the logical flow constraint is easily stated:

$$d_{12} + d_{35} \leq 1$$

(e) Consider running this program uninterrupted on a platform with a data cache. Assume that the data pointed to by x is not present in the cache at the start of this function.

For each read/write access to *x, argue whether it will be a cache hit or miss.

Now, assume that *x is present in the cache at the start of this function. Identify the basic blocks whose execution time will be impacted by this modified assumption.

Solution: The answer to this question depends on the path executed.

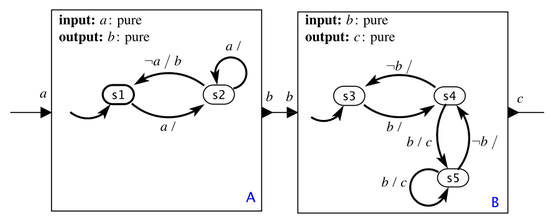
If the path is 1-3-4-5, then the access in Basic Block 3 is a miss, but the accesses in Block 4 will be a hit. The miss in Block 3 will occur even in path 1-3-5.

If the path is 1-2-1-3-4-5, then the access in Block 2 will be a miss, but subsequent acesses will hit.

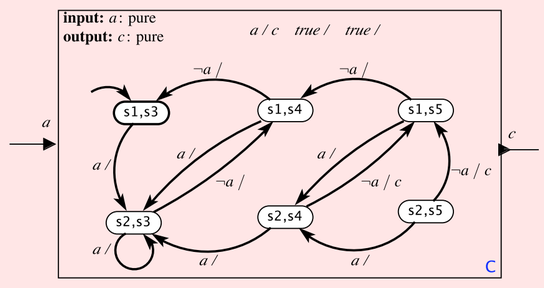
For the second part, if *x is present in the cache, then every access is likely to be a hit. The accesses in Blocks 2 and 3 are affected by this change.

p132, ex 3

3. Consider the following synchronous composition of two state machines *A* and *B*:



Construct a single state machine *C* representing the composition. Which states of the composition are unreachable?



The following states are unreachable: (s1,s5), (s2,s4), and (s2,s5).

p73, ex 7

7. Consider the state machine in Figure 3.15. State whether each of the following is a behavior for this machine. In each of the following, the ellipsis "$\cdots$" means that the last symbol is repeated forever. Also, for readability, *absent* is denoted by the shorthand a and *present* by the shorthand p.

(a) $x = (p, p, p, p, p, \cdots), \quad y = (0, 1, 1, 0, 0, \cdots)$
(b) $x = (p, p, p, p, p, \cdots), \quad y = (0, 1, 1, 0, a, \cdots)$
(c) $x = (a, p, a, p, a, \cdots), \quad y = (a, 1, a, 0, a, \cdots)$
(d) $x = (p, p, p, p, p, \cdots), \quad y = (0, 0, a, a, a, \cdots)$
(e) $x = (p, p, p, p, p, \cdots), \quad y = (0, a, 0, a, a, \cdots)$

input: x: pure
output: y: $\{0, 1\}$

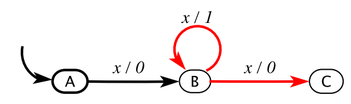


Figure 3.15: State machine for Exercise 7.

Scheduling

8 (5 p.)

Describe briefly Rate Monotonic Scheduling (RMS). The presentations should include the following parts:

a) describe the task model and execution assumptions,
b) the method to assign priorities to tasks, and
c) discussion whether RMS can provide optimal schedule and the condition for schedulability of tasks.

Draw a RMS schedule for the task set from Table 1.

L&S p333 or Marw. p312

Task	Period	WCET
1	6	1
2	12	3
3	24	5

Table 1: Task set for RMS scheduling.

Rate monotonic (RM) scheduling [348] is probably the most well-known scheduling algorithm for independent periodic tasks. Rate monotonic scheduling is based on the following assumptions ("**RM assumptions**"):

1. All tasks that have hard deadlines are periodic.
2. All tasks are independent.
3. $D_i = T_i$, for all tasks.
4. C_i is constant and is known for all tasks. Self-suspension (voluntarily relinquishing the execution) is not allowed.
5. The time required for context switching is negligible.
6. For a single processor and for n tasks, the accumulated utilization U_{sum} does not exceed the following bound:

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{6.7}$$

Figure 6.12 shows the bound of constraint (6.7).
The bound is about 0.7 for large n:

$$\lim_{n \to \infty} n * (2^{1/n} - 1) = log_e(2) = ln(2) \approx 0.7 \tag{6.8}$$

p257, ex1

Memory Architectures — Exercises

1. Consider the function compute_variance listed below, which computes the variance of integer numbers stored int the array data.

```
1  int data[N];
2
3  int compute_variance() {
4    int sum1 = 0, sum2 = 0, result;
5    int i;
6
7    for(i=0; i < N; i++) {
8      sum1 += data[i];
9    }
10   sum1 /= N;
11
12   for(i=0; i < N; i++) {
13     sum2 += data[i] * data[i];
14   }
15   sum2 /= N;
16
17   result = (sum2 - sum1*sum1);
18
19   return result;
20 }
```

Other courses focus on cache impact on code execution (e.g. effective C)

this is a micro-arch knowledge from another course (not taught here not checked here!)

Suppose this program is executing on a 32-bit processor with a direct-mapped cache with parameters $(m, S, E, B) = (32, 8, 1, 8)$. We make the following additional assumptions:

- An int is 4 bytes wide.
- sum1, sum2, result, and i are all stored in registers.
- data is stored in memory starting at address 0x0.

Answer the following questions:

(a) Consider the case where N is 16. How many cache misses will there be?

Solution: First, note that each block stored in the cache is of size 8 bytes. Since each int is 4 bytes wide, we note that data[0] and data[1] will lie in the same cache block, so will data[2] and data[3], and so on.

If N is 16, then we will suffer a cache miss in the first for loop on reading data[i] for every even i from 0 to 15. At the end of the first for loop, the entire array data will be in the cache. Thus, while executing the second for loop, we will never suffer a cache miss on any read.

Thus, the total number of cache misses is 8.

(b) Now suppose that N is 32. Recompute the number of cache misses.

Solution: If N is 32, data[i] and data[i+16] will map to the same cache block. Thus, during the first for loop, each read to data[i+16] will evict the block containing data[i].

Thus, when we execute the second for loop, we will suffer a cache miss on each even entry in the array all over again, just as in the first for loop.

Therefore, the total number of cache misses in this case will be 16*2 = 32.

(c) Now consider executing for N = 16 on a 2-way set-associative cache with parameters $(m, S, E, B) = (32, 8, 2, 4)$. In other words, the block size is halved, while there are two cache lines per set. How many cache misses would the code suffer?

Solution: The code would suffer 16 cache misses in the first for loop – one on reading each array entry. The reason the number of misses doubles is the smaller block size. In part (a), reading data[2*i] also moves data[2*i+1] into the cache, but this does not occur in the present case.

p132, ex 1

1. Consider the extended state machine model of Figure 3.8, the garage counter. Suppose that the garage has two distinct entrance and exit points. Construct a side-by-side concurrent composition of two counters that share a variable c that keeps track of the number of cars in the garage. Specify whether you are using synchronous or asynchronous composition, and define exactly the semantics of your composition by giving a single machine modeling the composition. If you choose synchronous semantics, explain what happens if the two machines simultaneously modify the shared variable. If you choose asynchronous composition, explain precisely which variant of asynchronous semantics you have chosen and why. Is your composition machine deterministic?

variable: c: $\{0, \cdots, M\}$
inputs: *up*, *down*: pure
output: *count*: $\{0, \cdots, M\}$

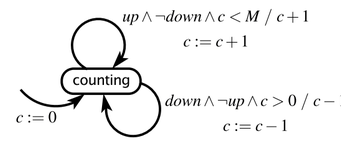
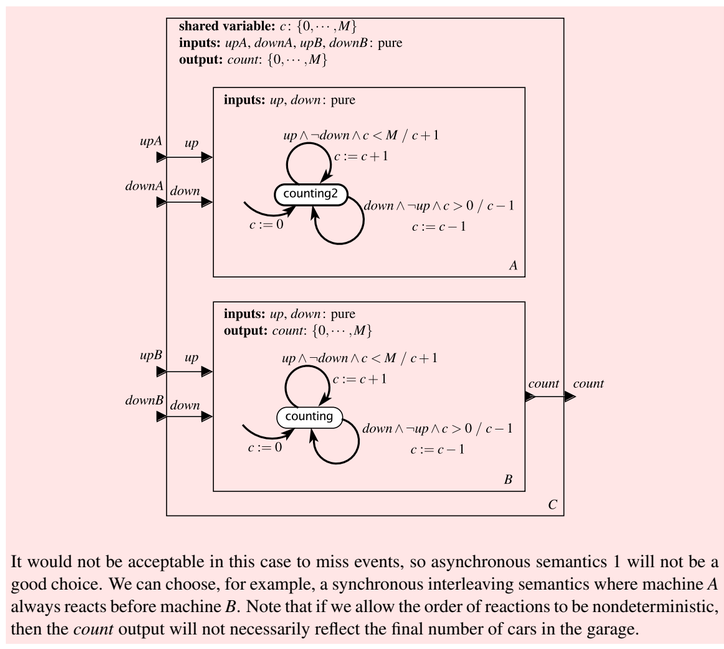


Figure 3.8: Extended state machine for the garage counter of Figure 3.4.



It would not be acceptable in this case to miss events, so asynchronous semantics 1 will not be a good choice. We can choose, for example, a synchronous interleaving semantics where machine *A* always reacts before machine *B*. Note that if we allow the order of reactions to be nondeterministic, then the *count* output will not necessarily reflect the final number of cars in the garage.

L&S p351

2. This problem studies dynamic-priority scheduling. Consider two tasks to be executed periodically on a single processor, where task 1 has period $p_1 = 4$ and task 2 has period $p_2 = 6$. Let the deadlines for each invocation of the tasks be the end of their period. That is, the first invocation of task 1 has deadline 4, the second invocation of task 1 has deadline 8, and so on.

(a) Let the execution time of task 1 be $e_1 = 1$. Find the maximum value for the execution time e_2 of task 2 such that EDF is feasible.

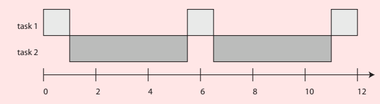
(b) For the value of e_2 that you found in part (a), compare the EDF schedule against the RM schedule from Exercise 1 (a). Which schedule has less preemption? Which schedule has better utilization?

(a) U <= 1; 1/4 + x/6 <= 1 … x = 9/2

2. This problem studies dynamic-priority scheduling. Consider two tasks to be executed periodically on a single processor, where task 1 has period $p_1 = 4$ and task 2 has period $p_2 = 6$. Let the deadlines for each invocation of the tasks be the end of their period. That is, the first invocation of task 1 has deadline 4, the second invocation of task 1 has deadline 8, and so on.

(a) Let the execution time of task 1 be $e_1 = 1$. Find the maximum value for the execution time e_2 of task 2 such that EDF is feasible.

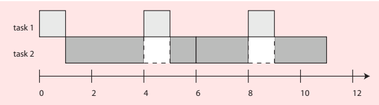
Solution: The largest execution time for task 2 is $e_2 = 4.5$. The following figure shows the resulting schedule:



The schedule repeats every 12 time units.

(b) For the value of e_2 that you found in part (a), compare the EDF schedule against the RM schedule from Exercise 1 (a). Which schedule has less preemption? Which schedule has better utilization?

Solution: Comparing the schedule in (a) with the schedule in Exercise 1(a), we see that EDF has no preemption at all, while RM performs two preemptions every 12 time units. Moreover, EDF has 100% utilization, whereas RM has less.



The schedule repeats every 12 time units.

7 Static Scheduling (4 p.) Exam 2021 april

Using list scheduling, make the schedule for the data dependency graph depicted in the figure below. You may use two adders and one pipelined multiplier. Adders have 1 clock cycle delay and the multiplier 2 clock cycles delay (two pipeline stages, 1 cycle for each pipeline stage). Answer the following questions:

a) How do you compute the priorities? Give the priority of each operation.
b) What is the number of clock cycles for execution of this model? Does this algorithm always finds the shortest possible schedule? Explain.

Give the sequence of steps taken by the list scheduling algorithm that leads to your solution. For each step give the list of nodes considered for scheduling.

Solution

List scheduling is addressed in Lecture 8. Several different priority functions are possible. The solution below uses the number of dependent nodes as priority. Priorities are as follows: 0:0, 1:2, 2:2, 3:4, 4:4, 5:1, 6:3, 7:0, 8:1, 9:0 With these in mind:

clk	add1	add2	mul	queues [+] [*]
0	3	2	4	[3,2,0] [4,1]
1	0		1	[0] [1]
2			6	[ ] [6]
3	5			[5] [ ]
4	8	7		[8,7] [ ]
5	9			[9] [ ]

6 clock cycles to finish.

L&S p207, ex2, Sensors

2. The dynamic range of human hearing is approximately 100 decibels. Assume that the smallest difference in sound levels that humans can effectively discern is a sound pressure of about 20 μPa (micropascals).

(a) Assuming a dynamic range of 100 decibels, what is the sound pressure of the loudest sound that humans can effectively discriminate?

(b) Assume a perfect microphone with a range that matches the human hearing range. What is the minimum number of bits that an ADC should have to match the dynamic range of human hearing?

7.1.3 Dynamic Range

Digital sensors are unable to distinguish between two closely-spaced values of the physical quantity. The **precision** p of a sensor is the smallest absolute difference between two values of a physical quantity whose sensor readings are distinguishable. The **dynamic range** $D \in \mathbb{R}_+$ of a digital sensor is the ratio

$$D = \frac{H - L}{p},$$

where H and L are the limits of the range in (7.2). Dynamic range is usually measured in decibels (see sidebar on page 189), as follows:

$$D_{dB} = 20 \log_{10} \left(\frac{H - L}{p} \right). \tag{7.3}$$

a) 100dB = 20 log (X/20microPa) … X = 10^5 20 10^{-6}

$$D_{dB} = 20 \log_{10} \left(\frac{H - L}{p} \right) 20 \log_{10}(2^n) = 20n \log_{10}(2) \approx 6n \; dB. \tag{7.4}$$

Each additional bit yields approximately 6 decibels of dynamic range.

b) n = ceil(100/6) … 17