Realtime System

General

Definition

Correctness and Execution time of the result are guaranteed. When a certain deadline is met.

- correct result
- specified time frame (meets a deadline)

Requirements

- Predictability
- Reliability
- minimal Delay/latency
- Correctness and Execution time of the result are guaranteed

2 most important deadlines

- correct result and have to meet deadlines

Three types of a RTS

(Andere Fragestellung: which are the different kinds of realtime systems, give one example each)

- Hard
    - missing a deadline is a total system failure (airbag)
- Soft
    - the usefulness of a result degrades after its deadline thereby degrading the system's quality of service(warning systems)
- Firm
    - Infrequent deadline misses are tolerable but may degrade the system‘s quality of service.
    - The usefulness of a result is zero after its deadline. (car - ignition point optimizer for motor)

Classification/Properties

Fragestellung: three criteria to classify realtime system

- centralised or distributed RTS
- interactive or autonomic systems
- hierarchical or independent system
- time-driven or event driven RTS
- consequences of missing deadline
- reliability and fault tolerance
- cyclic or asynchronous scheduling

Operating System

Operating system fulfil requirements

- small ticks(using normal seconds does not work in a RTS)
- Examples

Definition for:

- Relative Time
- Watch Dogs
    - Funktion anderer Komponenten überwacht. Wird dabei eine mögliche Fehlfunktion erkannt
- Alarm clock
- Timer
    - enable set the time start the time → gives a signal at the end of day

Main requirements for rtos

- timer with small ticks (ns)

Tasks

- if the task is non-interruptilble, it can have only 3 states:
    - ready (waiting to start)
    - running
    - completed

there is only one way to change states:

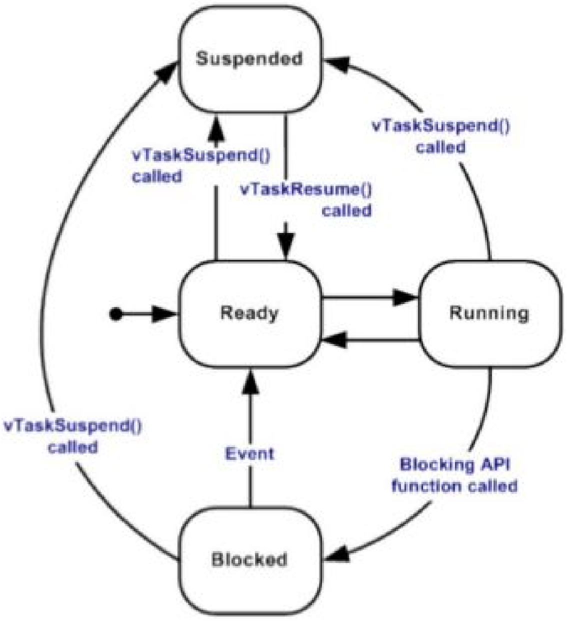
- ready (may be not necessary) → running → completed

when you use explicit plan there are only the states:

- running - completed

if the task is interruptible, there are 5 states:

- ready
- running
- blocked
- suspended
- completed



Running

When a task is actually executing it is said to be in the Running state. It is currently utilizing the processor. If the processor on which the OS is running only has a single core then there can only be one task in the Running state at any given time.

Ready

Ready tasks are those that are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.

Blocked

A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event. For example, if a task calls “sleep()” it will block (be placed into the Blocked state) until the delay period has expired - a temporal event. Tasks can also block to wait for queue, semaphore, event group, notification or semaphore event. Tasks in the Blocked state normally have a 'timeout' period, after which the task will be timeout, and be unblocked, even if the event the task was waiting for has not occurred. Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state

Suspended

Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded to do so through the “Suspend()” and “Resume()” API calls respectively.

Embedded Systems

- an embedded system is a computer that does not look a computer

Requirements of Embedded Systems

- at least one CPU
- has a physical context
- provides dedicated set of services to the end user
- does not provide general computing services to the end user
- often part of a larger environment
- have often real time constraints

Minor page fault/major page fault

- page faults cause an interrupt
  - if this happens, you need more memory
    - lack memory and forbid RTS from having page fault

Absolute vs. relative timestamps

Relativ:

- displays the number of minutes, hours, days, week or years ago a post was published

Absolute:

- display the exact date and time a post was published

(t0 changes absolute bad, relative still ok)??

Scheduling

- The Process of creating a schedule deciding how to order this tasks and how to commit resources between the variety of possible tasks

Feasible Schedule

- when each job can be completed with its individual deadline

EDF (earliest deadline first)

- has dynamic priorities, shorter deadline → higher priority
- executable task with high p will always interrupt a task with lower p

RMS(Rate Monotonic Scheduling)

- has a static priorities: shortest period → high p
- preemptive
- if a system not schedulable with RMS → can not be schedulable with any other static priority

EDD (earliest Due Date)

- non preemptive
- for periodic tasks with equal arrival time

Different between necessary test and sufficient Test:

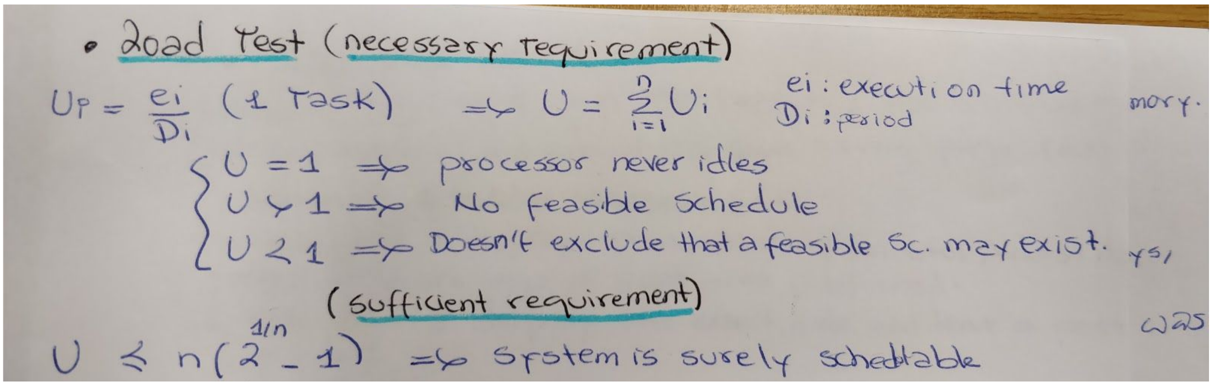
Necessary test (test fails = no feasible schedule):

- if one of the appropriate necessity test fails then there is no feasible schedule
- if one or more or all necessity test are fulfilled then there may be or may not be a feasible schedule

Sufficient Test (suff T. exists → feasible schedule):

- if you find at least (delta) suf. necessity test, than the task package is feasible schedulable
- if you can not find a suffice necessity text, than a feasible suf. may exist or not

Load Test (necessary Requirements):



• Load Test (necessary requirement)

$U_P = \frac{e_i}{D_i}$ (1 Task) ⟹ $U = \sum_{i=1}^{n} U_i$ ei : execution time   Di : period   mory.

$U = 1$ ⟹ processor never idles

$U > 1$ ⟹ No feasible Schedule

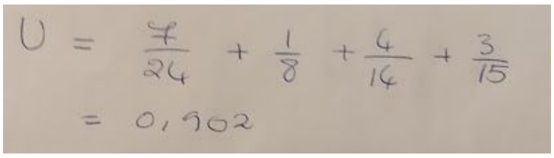
$U < 1$ ⟹ Doesn't exclude that a feasible Sc. may exist.   ys,

(sufficient requirement)   w25

$U \leq n(2^{1/n} - 1)$ ⟹ System is surely schedlable

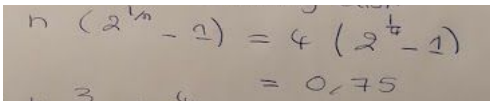
Beispiel:

T1(7,24), T2(1,8), T3(4,14), T4(3,15)   n=4 (anzahl der tasks)



$$U = \frac{7}{24} + \frac{1}{8} + \frac{4}{14} + \frac{3}{15} = 0,902$$

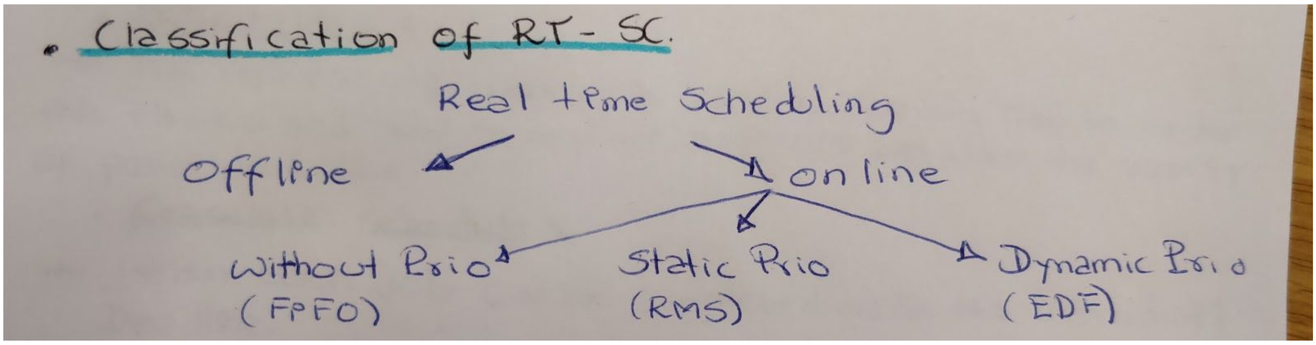
U<1 → doesn't exclude that a feasible scheduling exist



$$n(2^{1/n} - 1) = 4(2^{\frac{1}{4}} - 1) = 0,75$$

→ system is may schedulable

Classification of RT - SC.



Priority inversion:

- a high priority is blocked because a task of lower priority is using the resource

Difference between process and thread:

Process

- heavy unit of resource allocation and of protection
- process creation is very costly in terms of resources
- program execution a process is relatively slow
- process cannot access to memory belonging to another process
- process switching is time consuming
- process can contain several Threads

Thread:

- light weight unit of CPU utilisation
- thread creation is very economical
- program executing using threads are comparatively faster
- can access to memory area belonging to other threads within the same process

TCB(Thread Control block)

- thread state
- program counter
- CpU Registers
- CPU Scheduling info
- memory management info
- pointer to the process control block

PCB (Process Control Block)

- process state
- cpu registers
- cpu scheduling info
- accounting info
- I/O status info
- memory management info

Race Condition? how avoid?

- when 2 or more tasks access common data (memory)
- avoided by allowing access to critical selection for 1 task only at a time → mutual exclusion
- mathematical solution → semaphore

Mutex

a mutex is a special semaphore with n==1 (also called binary semaphore). it tells you only free of not and the queue respectively

Semaphore

semaphoren ist eine speicher sperre. mutex up → freigeben, mutex down→ sperren
mutex <0 → fehler

what issue do we have with concurrencies?

- Deadlock: one waits for another
- livelock: one triggers the other (distributed endless loop)
- race condition

Job

- a job is a single cpu-time requirement to perform a computational sequence

Execution Time

- the duration of a specific job between job request and job completion
- min net exc (ideal case)
- max net exc (worst case)

Sensor

- converts a physical or chemical measure into electrical signals

Assembling

- physical signal → sensor → low pass filter → sampling → quantization → coding

My quist Theorem

- sampling frequency : needs to be at least twice the maximum of the recorded signal(increase frequency)
- we need for that a low pass filter

if sample slowly

- higher freq. signals will be transferred

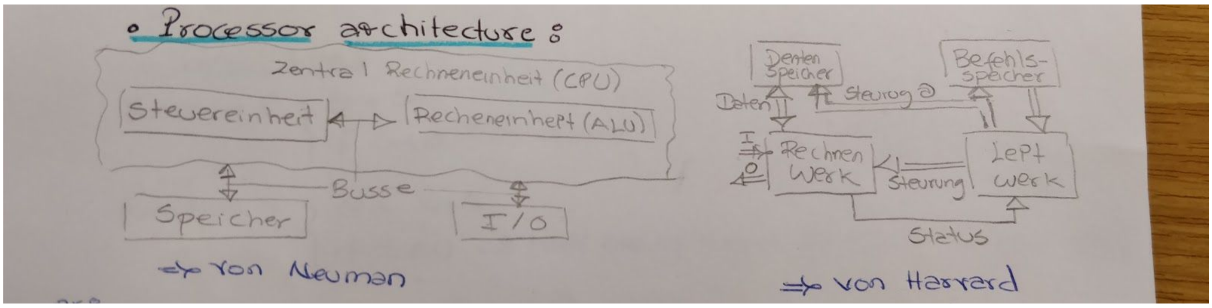
=>Aliasing:

- causes a false lower frequency component to appear in the sampled data of a signal

Anti Aliasing

- Remove all higher frequency signals

Processor Architecture



- ISP: Processors that come with a dedicated with a predefined Instruction set- similar CPU's like in PC's
- ASIP: are optimized in Hrd and software for the specific use of the system
- ASP: optimal performance due to complexity should only be done for processors that are needed for a single purpose (I/O process)

Microcontroller:

Development Process:

- develop it on pc
- cross compile it on pc
- deploy the executable onto the NC

Components:

- Timer:
- Memory
- Interface
- ADC (Analog Digital Converter)
- DAC (Digital Analog Converter)

Distributed RTS

- avoid collision on the Network line by connecting smaller parts with each other (segmentation) with only one connection

Petri Nets

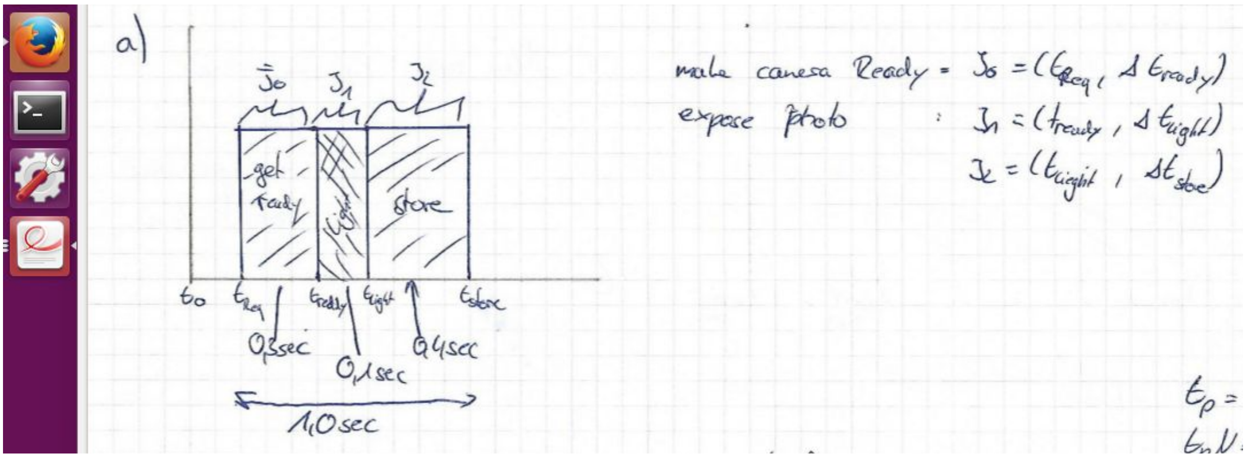
Definition

- simple process model with:
    - 3 elements: places, transitions and arcs (+token)
    - graphical and mathematical description
    - formal semantics and allows for analysis
- Enabled transmission
    - if each of its input contains at least one token
    - an enabled token can fire
        - when it fires it consumes a token from each input and places it in each output
- liveness:
    - a transition is live if it can never deadlock
- safeness:
    - the number of tokens in a place never executes one

(Für dominik: safe bedeutet, dass die anzahl der tokens in den knoten niemals als 1 überschreiten)

Praxis

Scheduling:



make camera Ready = $J_0 = (t_{Req}, \Delta t_{ready})$

expose photo : $J_1 = (t_{ready}, \Delta t_{light})$

$J_2 = (t_{light}, \Delta t_{store})$

$t_p =$

$t_n V$

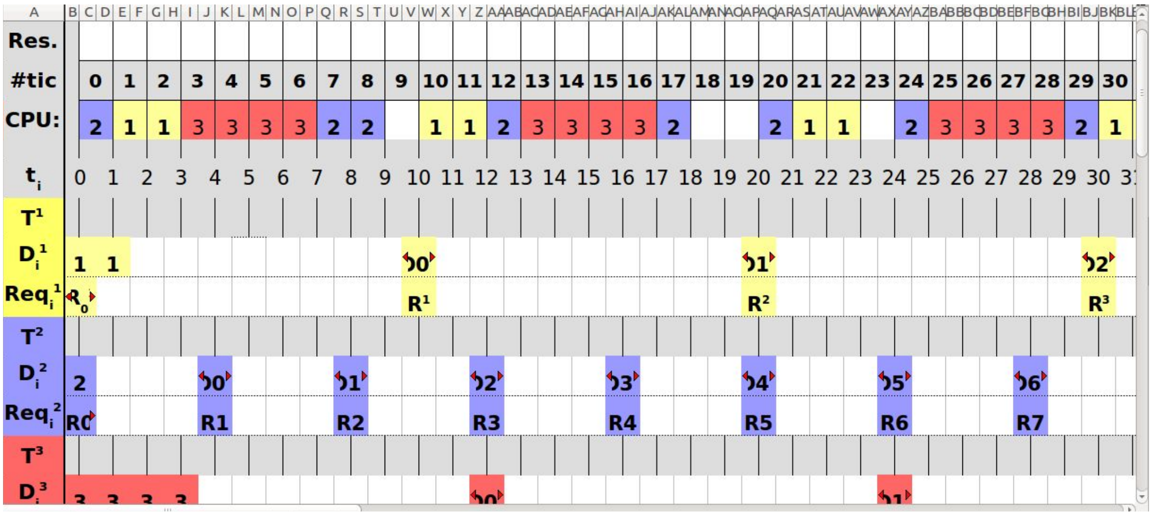
Rate berechnen:

Formel:

- rate = 1/(delta) T   T = periode

EDD (earliest due date)

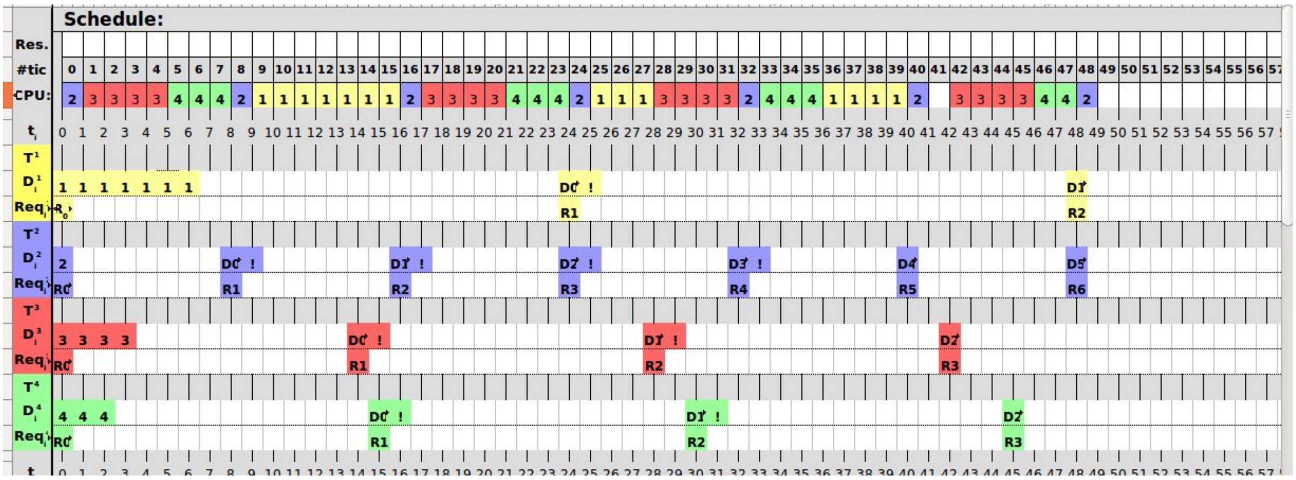
T1(2, 10), T2(1, 4), T3(4, 12)



Res.																															
#tic	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
CPU:	2	1	1	3	3	3	3	2	2		1	1	2	3	3	3	3	2			2	1	1		2	3	3	3	3	2	1
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
T^1																															
D_i^1	1	1									ↄ0										ↄ1										ↄ2
Req_i^1	R_0										R^1										R^2										R^3
T^2																															
D_i^2	2				ↄ0				ↄ1				ↄ2				ↄ3				ↄ4				ↄ5				ↄ6		
Req_i^2	R0				R1				R2				R3				R4				R5				R6				R7		
T^3																															
D_i^3	3	3	3	3									ↄ0												ↄ1						

EDF (earliest deadline first)

T1(7, 24) , T2(1, 8), T3(4, 14) , T4(3,15)



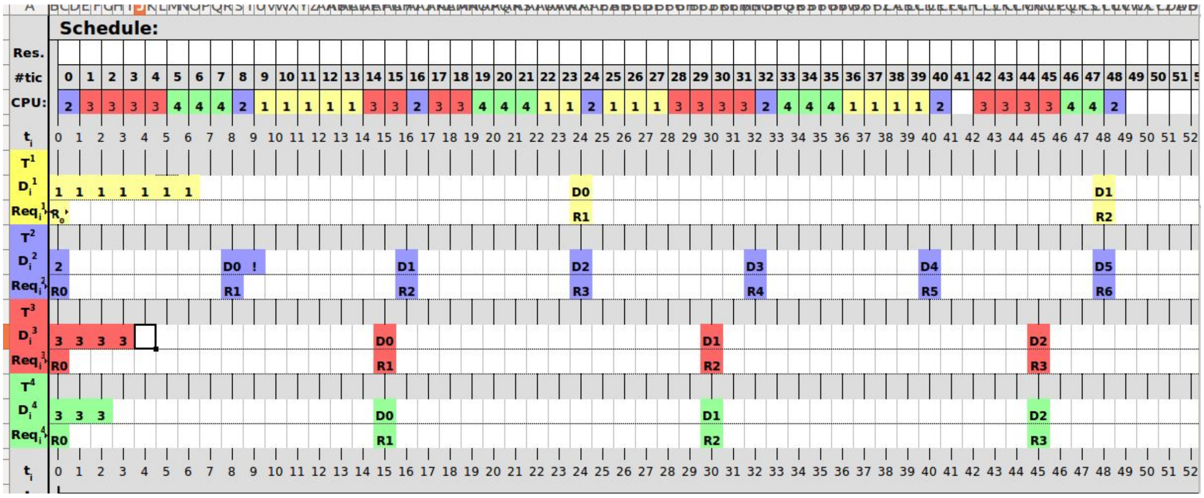
Schedule:

Res.																													
#tic	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
CPU:	2	3	3	3	3	4	4	4	2	1	1	1	1	1	1	1	2	3	3	3	3	4	4	4	2	1	1	1	3
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
T^1																													
D_i^1	1	1	1	1	1	1	1																		D0	!			
Req_i^1	R_0																								R1				
T^2																													
D_i^2	2								D0	!							D1	!							D2	!			
Req_i^2	R0								R1								R2								R3				
T^3																													
D_i^3	3	3	3	3											D0	!													D1
Req_i^3	R0														R1														R2
T^4																													
D_i^4	4	4	4													D0	!												
Req_i^4	R0															R1													
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Res.																													
#tic	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
CPU:	3	3	3	2	4	4	4	1	1	1	1	2		3	3	3	3	4	4	2									
t_i	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
T^1																													
D_i^1																				D1									
Req_i^1																				R2									
T^2																													
D_i^2				D3	!							D4								D5									
Req_i^2				R4								R5								R6									
T^3																													
D_i^3	!													D2															
Req_i^3														R3															
T^4																													
D_i^4		D1	!														D2												
Req_i^4		R2															R3												
t_i	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57

RMS (rate monotonic scheduling)

T1(7, 24) , T2(1, 8), T3(4, 14) , T4(3,15)



Schedule:

Res.																											
#tic	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
CPU:	2	3	3	3	3	4	4	4	2	1	1	1	1	1	3	3	2	3	3	4	4	4	1	1	2	1	1
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
T^1																											
D_i^1	1	1	1	1	1	1	1																		D0		
Req_i^1	R_0																								R1		
T^2																											
D_i^2	2								D0	!							D1								D2		
Req_i^2	R0								R1								R2								R3		
T^3																											
D_i^3	3	3	3	3												D0											
Req_i^3	R0															R1											
T^4																											
D_i^4	3	3	3													D0											
Req_i^4	R0															R1											
t_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Res.																										
#tic	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	5
CPU:	1	3	3	3	3	2	4	4	4	1	1	1	1	2		3	3	3	3	4	4	2				
t_i	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
T^1																										
D_i^1																						D1				
Req_i^1																						R2				
T^2																										
D_i^2						D3								D4								D5				
Req_i^2						R4								R5								R6				
T^3																										
D_i^3				D1															D2							
Req_i^3				R2															R3							
T^4																										
D_i^4				D1															D2							
Req_i^4				R2															R3							
t_i	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52

Petri Nets Examples

Exercise 7: Petri Net (optional): traffic lights

Try the examples in PIPE. Develop a Petri-Net for simple traffic lights:

a) for pedestrians,

b) for cars (replace the German state "red-yellow" by "yellow".

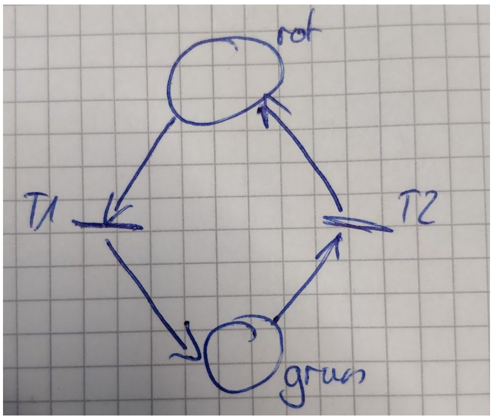
c) Combine a) and b).

d) When you feel comfortable you may

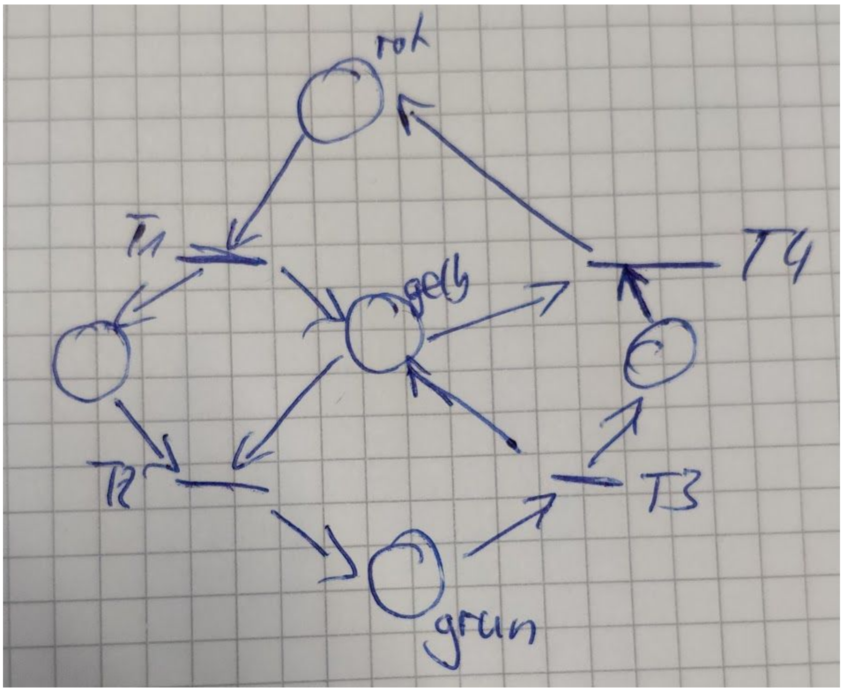
develop a real existing traffic crossing in Frankfurt

(avoid the big ones!)

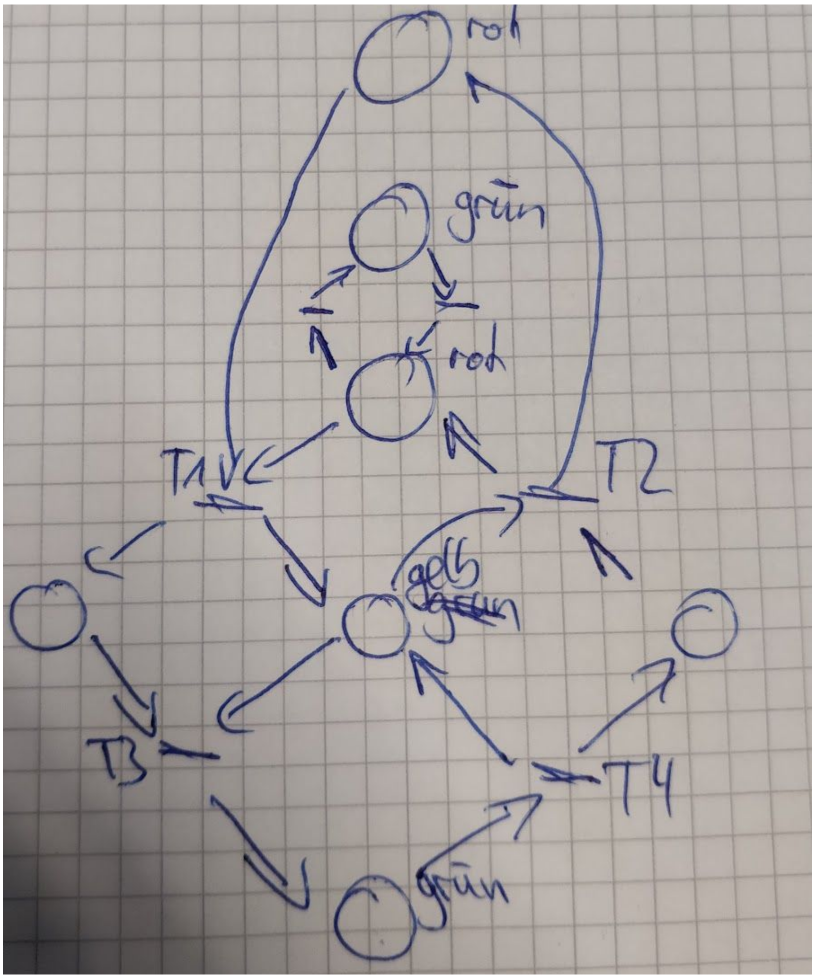
a)



b)



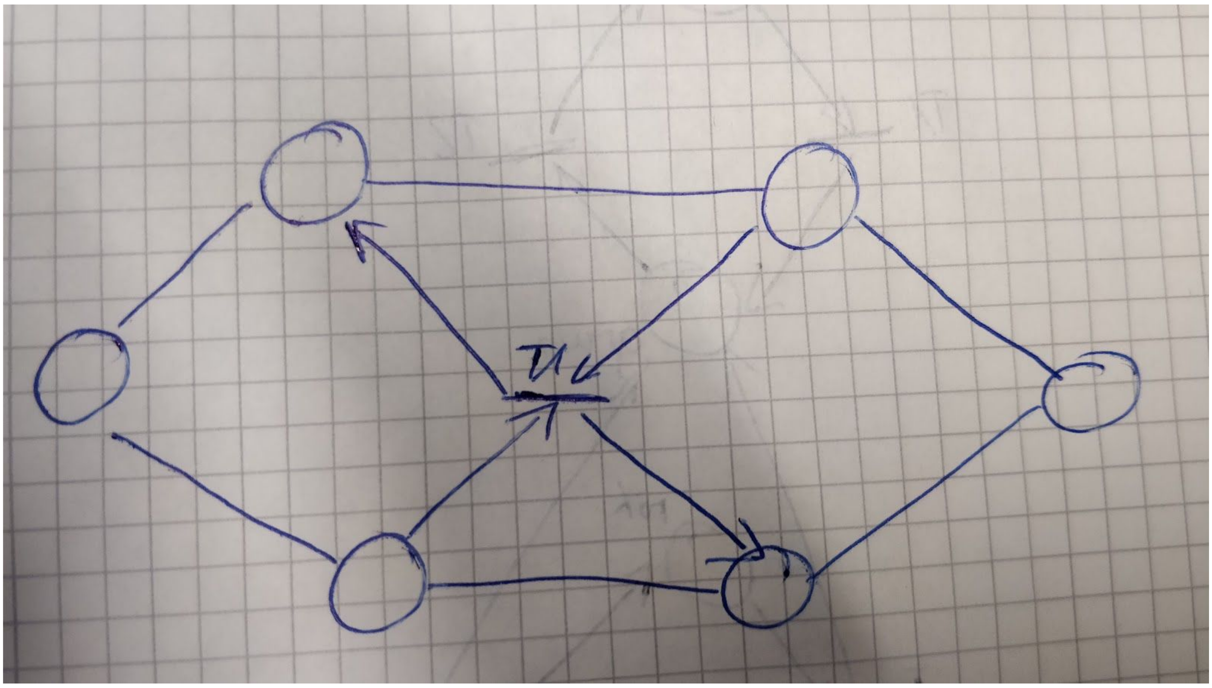
c)



Exercise 6: Petri Net (optional): railways

Imagine a railway line with two tracks to two directions between two stations. In the middle part there is only one track available. If you can't imagine the situation, take a tour to with tramway no 17 in Frankfurt and find the situation in place between station "Varrentrappstraße" and "Nauheimerstraße" near Messe.

You need not to model the traffic lights. However, if you feel comfortable with your solution you may add the traffic lights

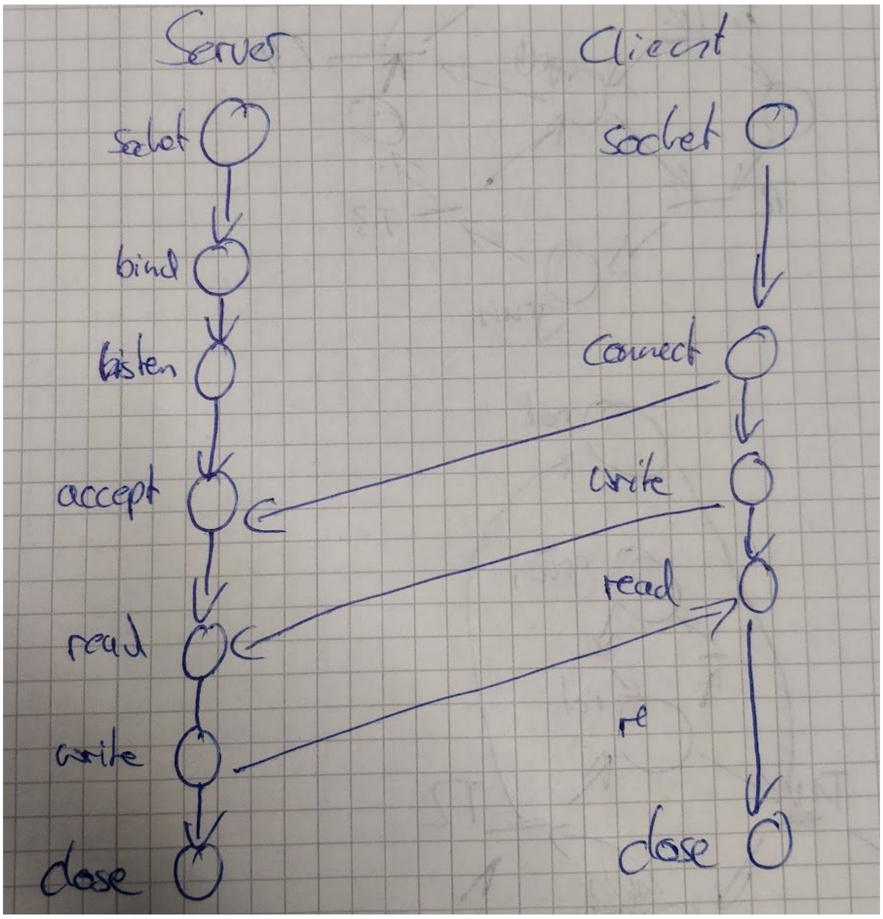


Exercise 9: Petri Net (optional): TCP/IP

Try to develop the transmission of a request/response pattern of TCP/IP

(this is part of the lecture "Networks" or "Distributed Systems").

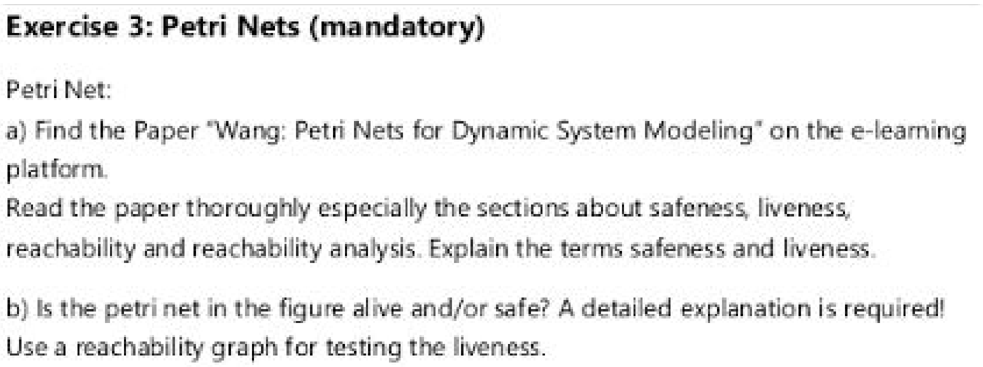
You may use a colored Petri-Net in PIPE, but it is not necessary.



Reachability Graph:

https://www.youtube.com/watch?v=EhSw-EpKjxM

Example 1:

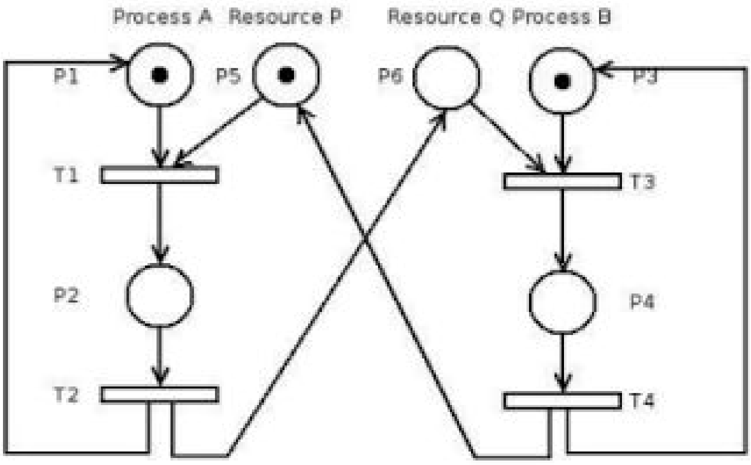


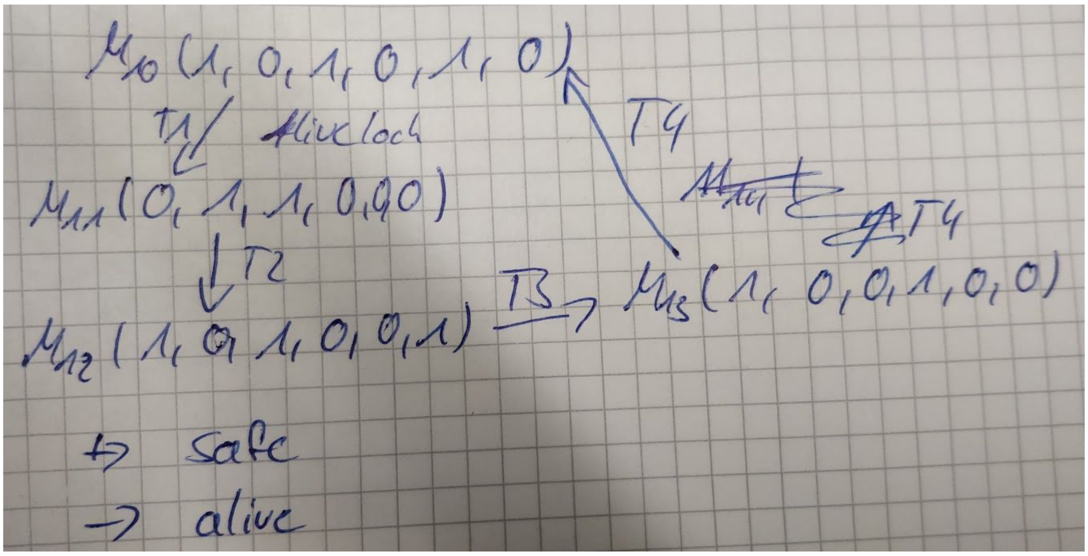
Exercise 3: Petri Nets (mandatory)

Petri Net:

a) Find the Paper "Wang: Petri Nets for Dynamic System Modeling" on the e-learning platform.
Read the paper thoroughly especially the sections about safeness, liveness, reachability and reachability analysis. Explain the terms safeness and liveness.

b) Is the petri net in the figure alive and/or safe? A detailed explanation is required!
Use a reachability graph for testing the liveness.

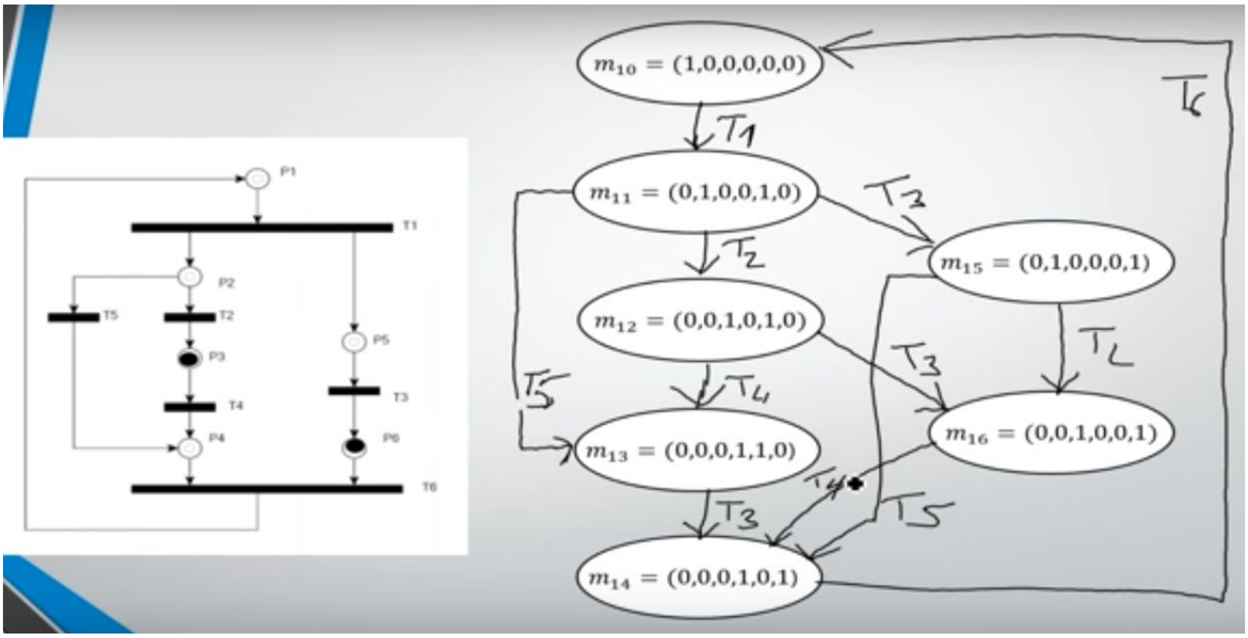




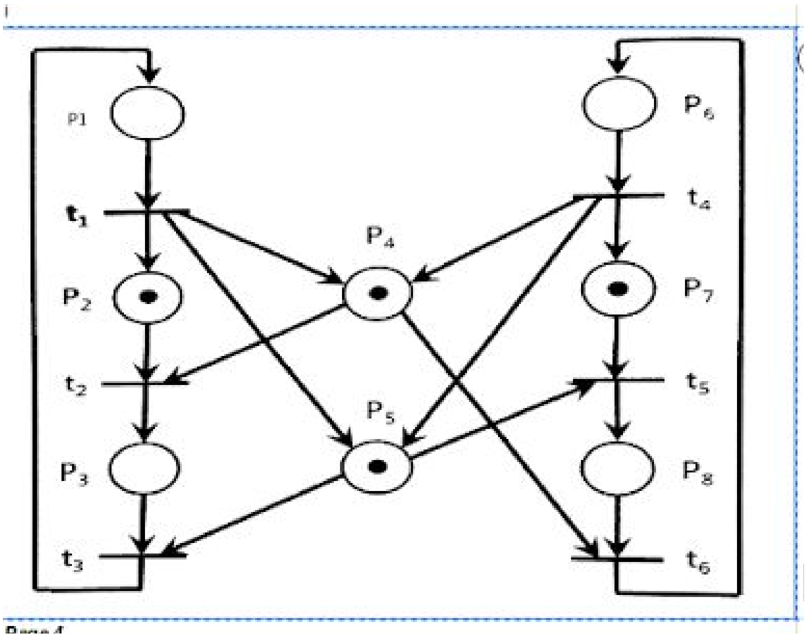
→ safe

→ alive

Example 2:

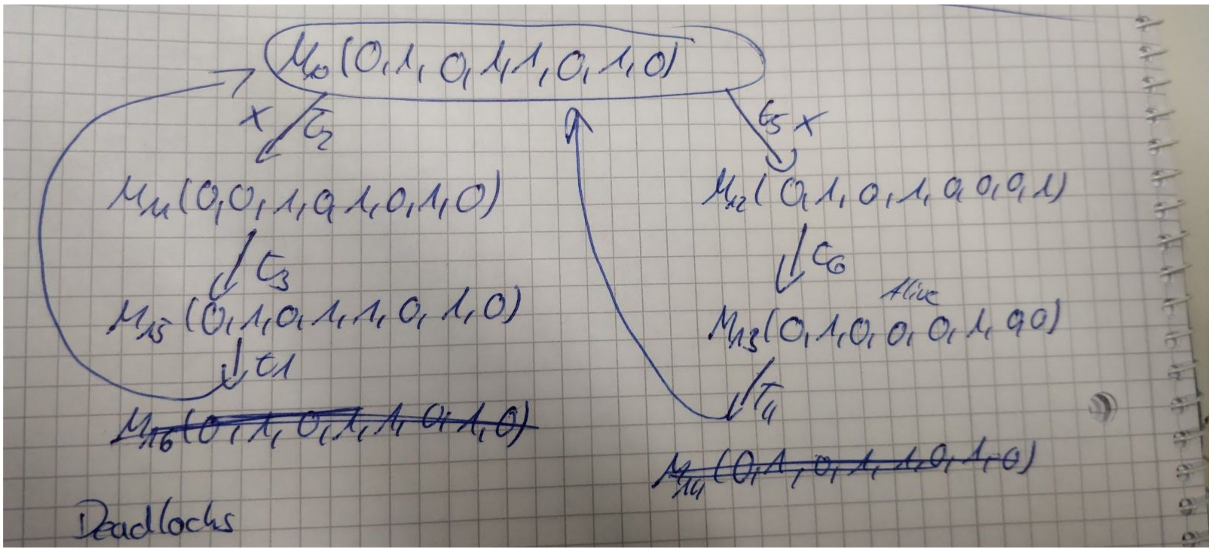


Example 3



Ab und zu entsteht ein livelock, das bedeutet wenn man mit p7 + p5→ p8 geht und dann p8+p4 in kreis fährt entsteht bei p2 ein livelock. Da man mit p7+p5-->p8 geht und gleichzeitig mit p2+p4 → p3 geht ensteht ein deadlock

Daraus folgt es ist nicht liveness
aber es ist safe



Link zu Klausur-Lösung
https://docs.google.com/spreadsheets/d/1n-HOtduOd0PoJERxK6E1u9Fx8dFCtRUEwWIAgx18CKE/edit?ts=5a2151e3#gid=0