Software Modeling and Verification
Lehrstuhl für Informatik 2
RWTH Aachen University

Prof. Dr. Ir. J.-P. Katoen
C. Kern

Foundations of the UML

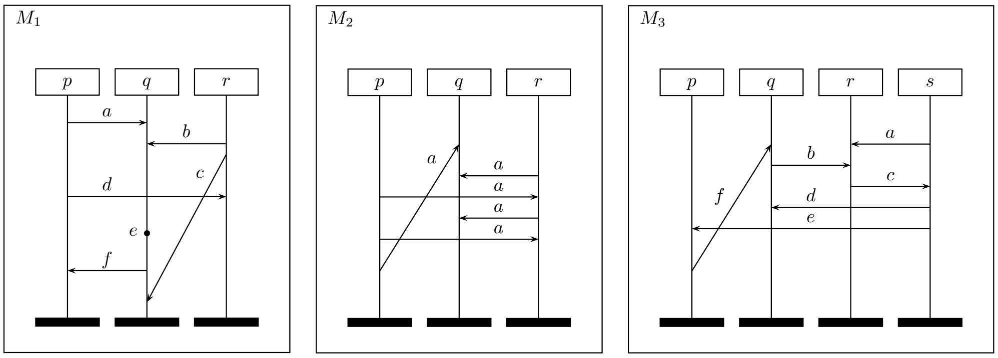
Winter Term 07/08

– Assignment 1 –

Hand in until November 07th before the exercise class.

Exercise 1 (5 points)

Let the following pictures M_1, M_2, M_3 be given:

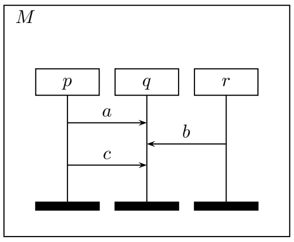


a) Write down the formal description of MSC M_1 as it was presented in the lecture.

b) Prove or disprove that M_2 and M_3 are MSCs.

Exercise 2 (5 points)

Determine all linearizations of the following MSC M:



Exercise 3 **(10 points)**

In this exercise we consider words over sending and receiving actions, only (i.e., there are no local actions). Write down a pseudo-code function that, given a word `w` $\in Act^*$, determines whether `w` is a linearization of an MSC. If `w` is not a linearization of an MSC the algorithm has to terminate at the first location where a contradiction to an MSC linearization occurs. The header of the function to implement looks as follows:

```
boolean isMSCLinearization(Act[] w)
```

Use the following methods to ease your work:

<u>Class ChannelSystem:</u>

A ChannelSystem is a collection of channels.

```
ChannelSystem(Process from, Process to)
     //constructor for an empty channel system
boolean addChannel(Process from, Process to)
     //creates a new channel (from,to) (if it does not exist, yet) and
     //returns true iff new channel was created
void putToChannelEnd(Process from, Process to, Message m)
     //appends m to channel (from,to) if channel exists
Message lookAtChannelHead(Process from, Process to)
     //peeks at head of channel without removing the element and returns message
     //content of head element
void removeFromChannelHead(Process from, Process to)
     //removes the element at the head of buffer (from,to)
boolean allChannelsEmpty()
     //returns true iff all channels within the channel system are empty
boolean channelExists(Process from, Process to)
     //returns true iff channel (from,to) exists
```

<u>Class *Act*:</u>

```
boolean isSending()
     //returns true iff this action is of type sending
boolean isReceiving()
     //returns true iff this action is of type receiving
Process getSendingProcess()
     //returns the sending process of this action
Process getReceivingProcess()
     //returns the receiving process of this action
Message getMessage()
     //returns the message content of this action
```

<u>Class Message:</u>

```
boolean equals(Message m)
     //returns whether this message is equal to m
```

Exercise 4 **(5 points)**

As presented in the second lecture, the *weak concatenation* of two MSCs M_1 and M_2 (with $M_i = \langle \mathcal{P}_i, E_i, \mathcal{C}_i, \ell_i, m_i, <_i \rangle$ for $i \in \{1, 2\}$) intuitively is realized by gluing the process lines together such that M_1 is situated on top of MSC M_2 (cf. Figure 1).

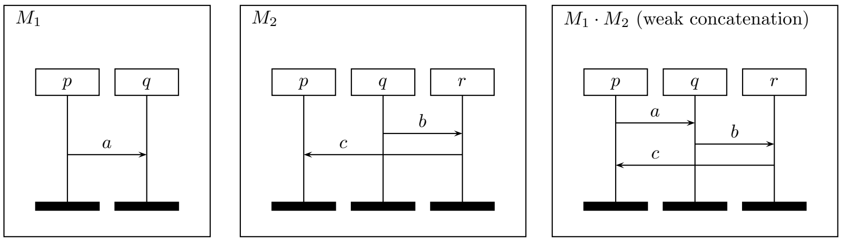


Figure 1: Two MSCs and their weak concatenation

Define the so-called *strong concatenation* $\cdot_s$ of two MSCs M_1 and M_2, i.e., all events of MSC M_1 have to be executed before the first event of M_2. For this purpose determine a structure $M = M_1 \cdot_s M_2 = \langle \mathcal{P}, E, \mathcal{C}, \ell, m, < \rangle$, that (in terms of M_1 and M_2) results from concatenating the two MSCs strongly.

Exercise 5 **(10 points)**

Formally prove or disprove the correctness of the following statements for i) MSGs (i.e., $M_i \in \mathbb{MSC}$, $i \in \{1, 2, 3\}$) and ii) CMSGs (i.e., $M_i \in \mathbb{CMSC}$, $i \in \{1, 2, 3\}$):
(remember: $|$ $\widehat{=}$ choice, $\times$ $\widehat{=}$ (weak) sequence, * $\widehat{=}$ iteration)

a) $M_1 | M_2 = M_2 | M_1$

b) $M_1 \times M_2 = M_2 \times M_1$

c) $(M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3)$

d) $(M_1 | M_2) | M_3 = M_1 | (M_2 | M_3)$

e) $(M_1 \times M_2) | M_3 = (M_1 | M_3) \times (M_2 | M_3)$

f) $(M_1 | M_2) \times M_3 = (M_1 \times M_3) | (M_2 \times M_3)$

g) $M_1^* | M_2^* = (M_1 | M_2)^*$