Note:

- Solution have to be handed in in groups of two.

Exercise 1 (Prove or disprove): (5 Points)

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

1. $M_1|M_2 = M_2|M_1$
2. $(M_1|M_2)|M_3 = M_1|(M_2|M_3)$
3. $(M_1 \bullet M_2)|M_3 = (M_1|M_3) \bullet (M_2|M_3)$
4. $(M_1|M_2) \bullet M_3 = (M_1 \bullet M_3)|(M_2 \bullet M_3)$
5. $M_1^*|M_2^* = (M_1|M_2)^*$

Exercise 2 (Linearisation): (8 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:

```
public static 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
```

Class Message:

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

Exercise 3 (CMSGs Properties): (6+6+3 Points)

Consider the (C)MSGs from figure 1 (last page):

1. Prove or disprove the following properties for the MSGs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$:
   a) local-choice (as defined in the lecture)
   b) regularity (as defined in Definition 1 at the end of this assignment)
2. Prove or disprove the following property for the CMSG $\mathcal{G}_4$:
   a) safety (as defined in Definition 2 at the end of this assignment)

In each case justify your answer in detail. If there are several reasons why a property does not hold, state at least two of them.

Definition 1: A Message Sequence Graph $\mathcal{G}$ is *regular* if each MSC labeling a loop in $\mathcal{G}$ has a strongly connected communication graph.

Definition 2: A compositional Message Sequence Graph $\mathcal{G}$ is called *safe* if every sequence of CMSCs (using the concatenation defined in the lecture) describing an accepting path of $\mathcal{G}$ results in an MSC.

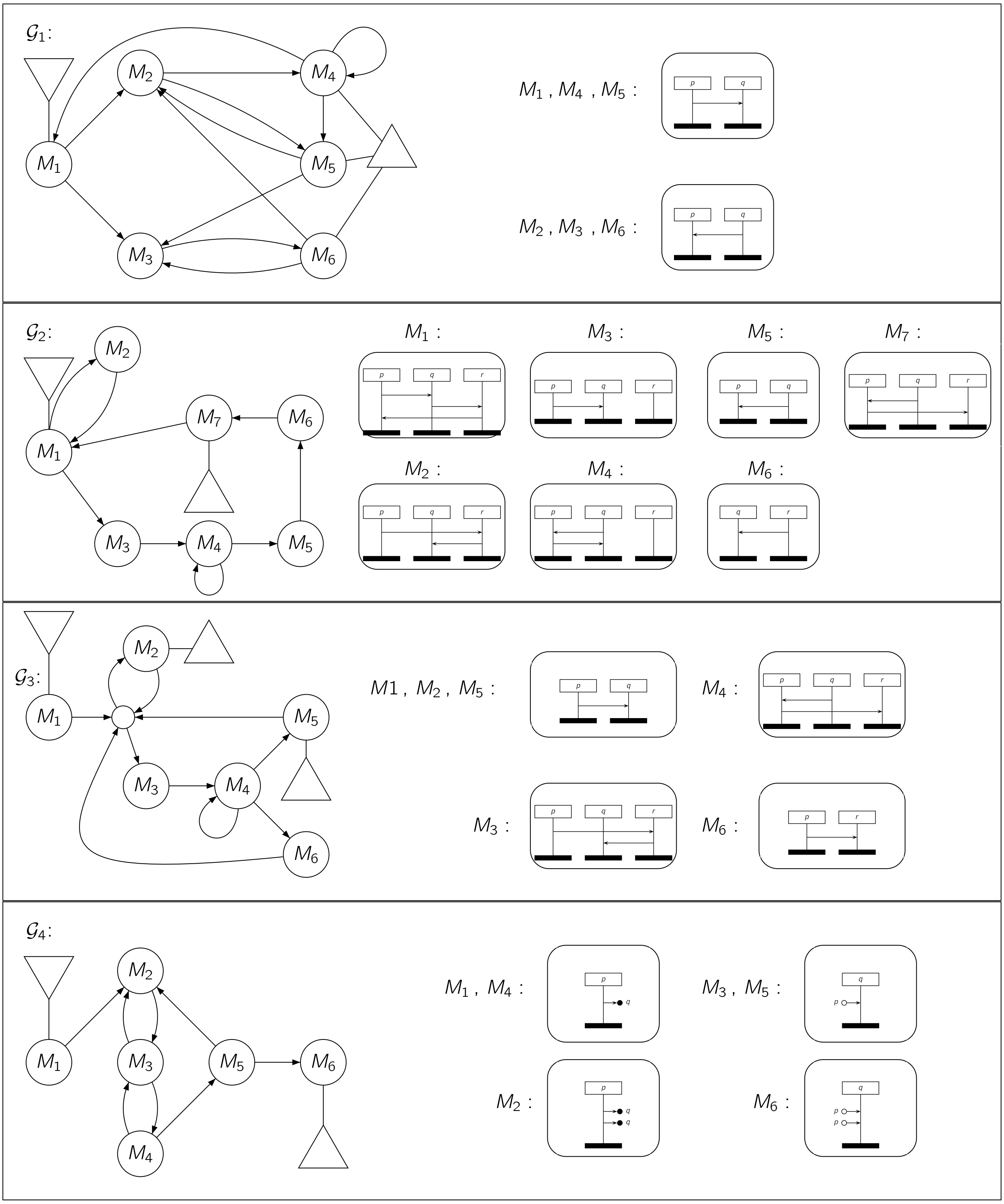

Abbildung 1: (C)MSGs for exercise 3