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

– Lecture number 4 –

(Date (19th Nov 2007))
summarized by *Muhammad Ali(284152)* and *Mudassir Rasool(284168)*

Message Passing Automata

The lecture was about Message Passing Automata and it consisted of the following sections...

- What is a Message Passing Automaton (will be abbreviated as MPA)?
- Formal Definition of MPA
- How MPA work (with the help of Illustration)?
- Configurations and Linearizations of MPAs
- Undecidability of non emptiness in MPAs
- Subclasses of MPAs, Boundedness

We will now summarize each of these sections.

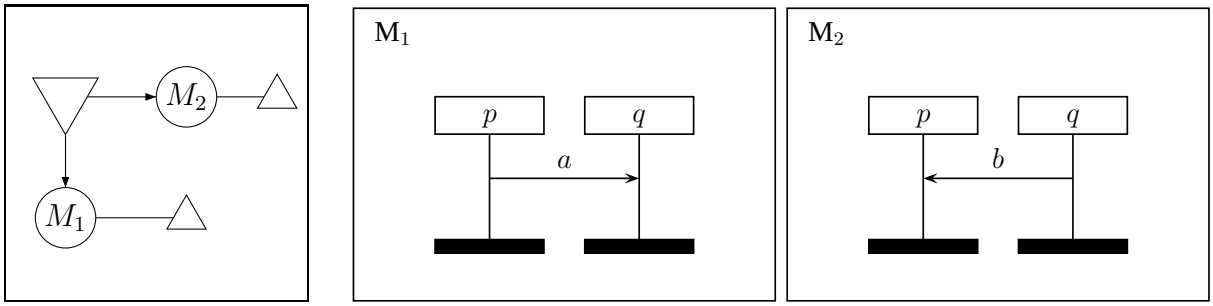
1 What is a Message Passing Automaton?

1.1 Definition

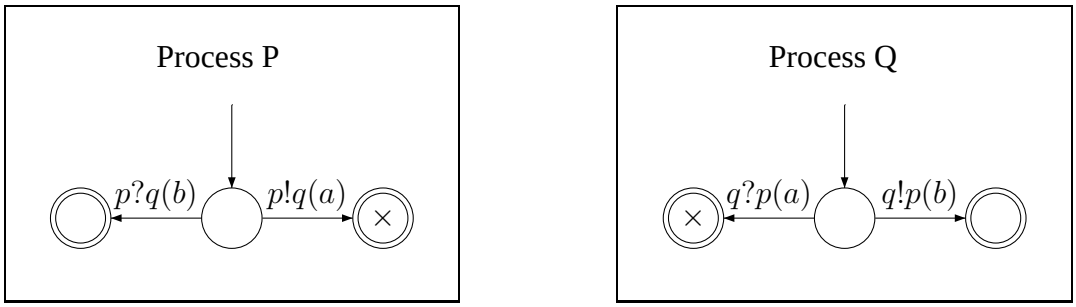
Message Passing Automata are "finite set of finite state automata plus communication channels (FIFO channels)". They are used to realize or exhibit the behavior/specification /scenerios of systems modelled by a (C)MSG".

1.2 Example Illustration

Suppose we want to realize the following MSG...

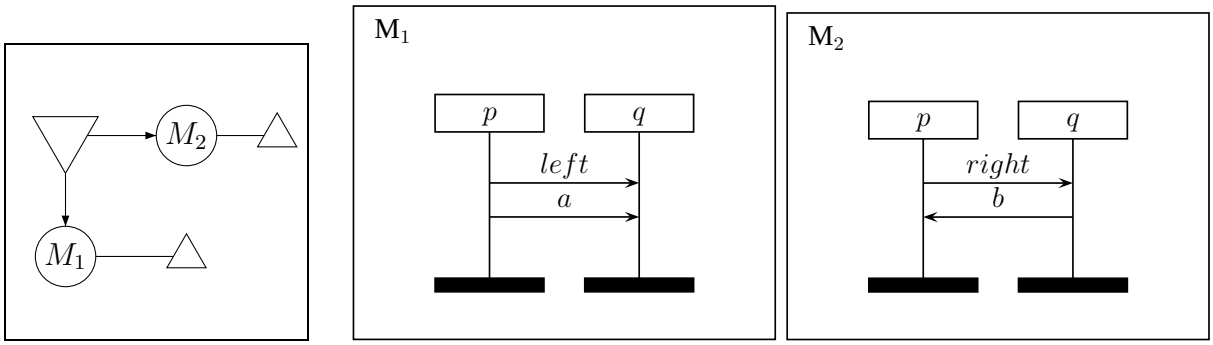


The realization of the above specification (that is MPA) would yield us the following...

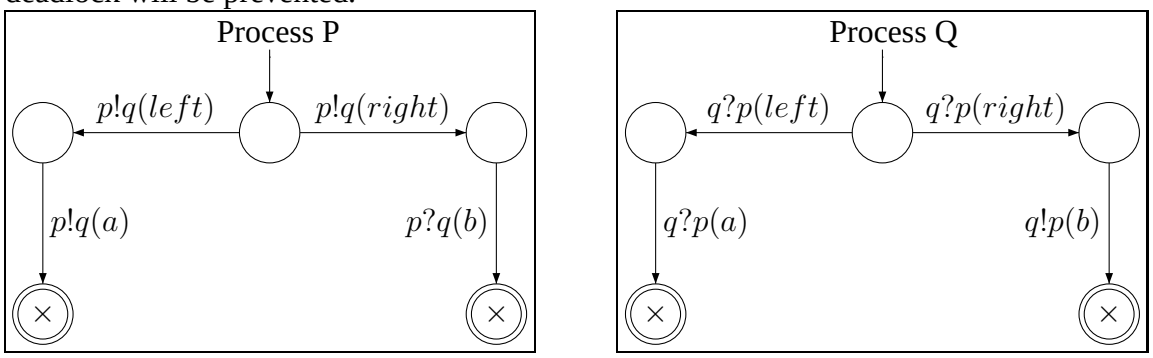


Because of the non-deterministic choice in the MSG, the process P (and Q) may either behave like MSC M_1 or M_2. But there is a possibility that process P may behave like M_1 and process Q may behave like M_2(or vice versa). The MPA for both the processes above shows this idea.

For instance, in left figure, process P may send a message to process Q but at the same time may also be waiting to receive a message from Q. To avoid such a deadlock, control messages can be added. The next picture will show the possible MSG after the addition of control messages.



The control messages "left" and "right", as shown above, resolve the problem and the deadlock will be prevented.



The intuition behind is to allow only one process to decide/dictate the execution through any (non deterministic) path with in any MSG. That single process should always contain the "minimal event" on all the paths of an MSG. In our case above it is process P.

2 Formal Definition of MPA

Let

- P = a finite set of at least two sequential processes.
- C = a finite set of message contents.

Then a message-passing automaton (MPA) A over P and C is a structure defined as...

$$A = (((S_p, \triangle_p))_{p \in P} , D , s_{init} , F)$$

Where

- D is a nonempty finite set of synchronization messages (or data).
- for each p $\in$ P
  - S_p is a nonempty finite set of local states (the S_p are disjoint).
  - $\triangle_p \subseteq S_p \times Act_p \times D \times S_p$ is a set of local transitions.
- $s_{init} \in S_A$ is the global initial state.
- $F \subseteq S_A$ is the set of global final states.

Where

$S_A = \prod_{p \in P} S_p$ is the set of global states of A.

From the definition it is clear that an MPA defines the collective behavior of all the individual automata in the system, each corresponding to some process p $\in$ P, and their mutual interaction. It also identifies the DEADLOCK conditions which may arise by the interaction of all the processes (or their individual automaton).

A system must have at least two processes and they should be interacting with each other to construct an MPA. In the other situation, if there is a single process or multiple independent processes running together there is no point to develop an MPA.

Let's now look at an example execution of an MPA in the next section...

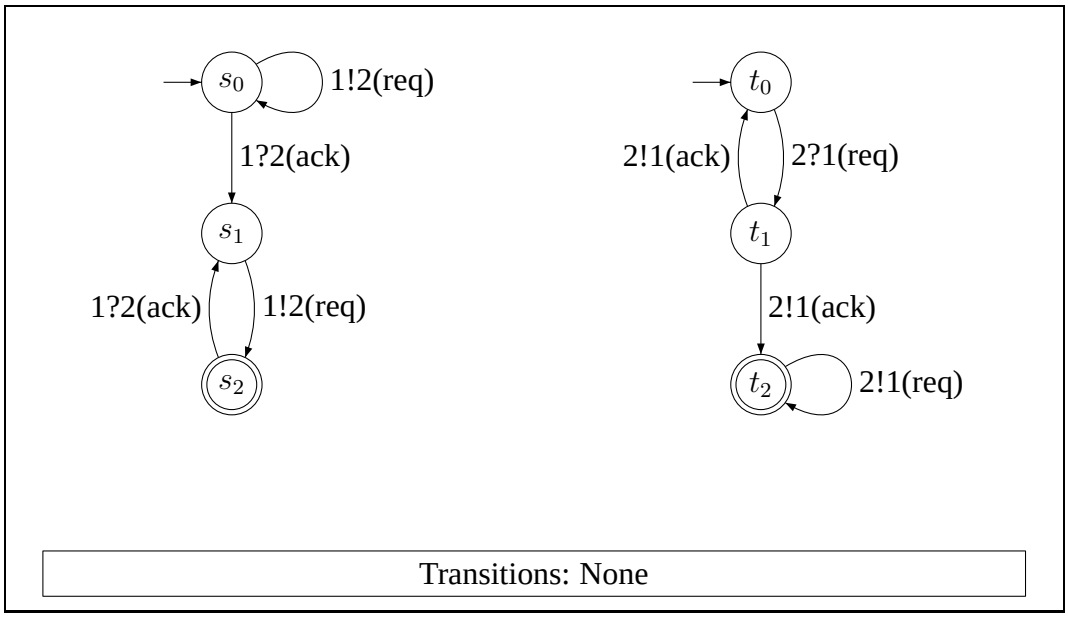
3 How MPA work (with the help of Illustration)?

Suppose we have an MPA A over P = {1,2} and C = {req,ack} and other arguments are:

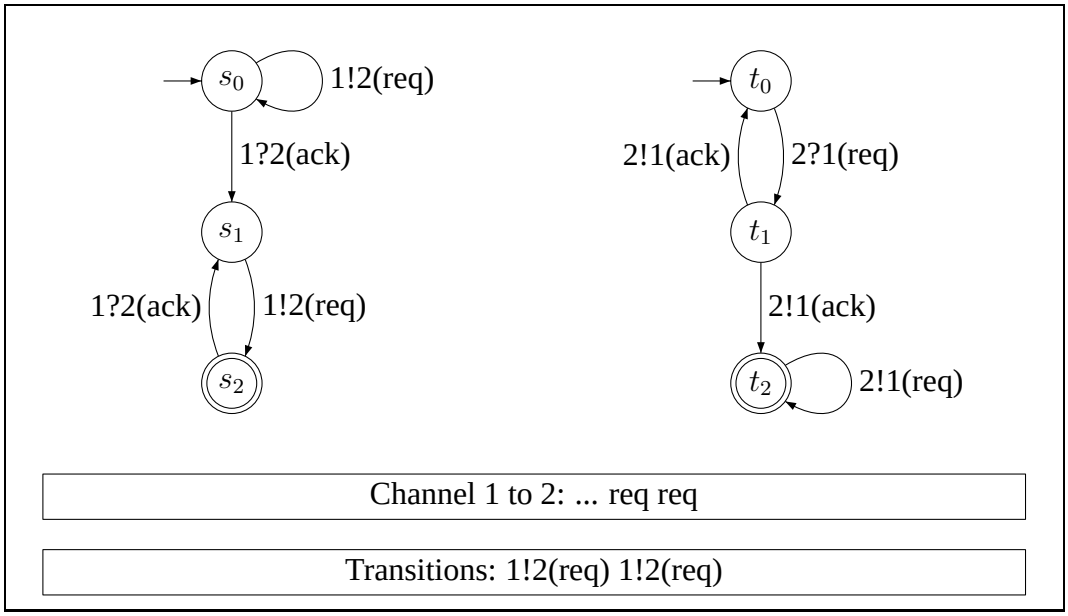
- S_1 = set of states for process 1 = {s_0,s_1,s_2}
- S_2 = set of states for process 2 ={t_0,t_1,t_2}
- $\triangle_1$ = set of local transitions for process 1: $s_0 \xrightarrow{1!2(req)_1} s_0$...
- $\triangle_2$ = set of local transitions for process 2: $t_0 \xrightarrow{2!1(req)_2} t_1$...
- s_{init} = set of initial states($S_{I1} \times S_{I2}$) = (s_0,t_0)
- F = set of final states ($S_{F1} \times S_{F2}$) = {(s_2,t_2)}

The system is at the initial states (s_0, t_0), Process 1 is at s_0 and Process 2 is at t_0 .

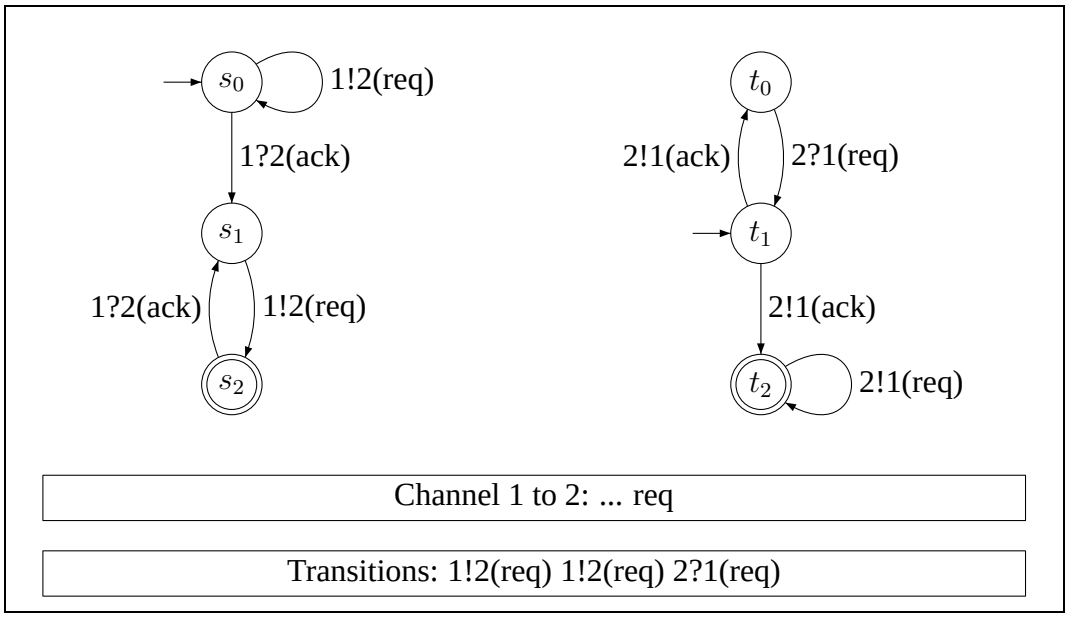
The following figure depicts the initial state of the system.



At this state, the process 1 will keep on sending sending the "req" messages and will remain in the same s_0 state. So after two such messages, there will be two "req" messages in the channel buffer. The picture below shows this state, the messages are not currently read by process 2.



As soon as the first "req" message is read by the process 2, it will move to state t_1. This state is shown in the picture on the following page.



In the next possible scenerio, process 2 may send "ack" message to process 1 and go state t_0.

The MPA will keep on executing this way till both the processes get to their final states s_2 and t_2 respectively, also all the channels should be empty too. If we get to such a condition then it will be an accepting run.

One of the possible sequences of transitions when this automaton will run (an accepting run) is as follows.....

1!2(req) 1!2(req) 2?1(req) 2!1(ack) 2?1(req) 2!1(ack) 1?2(ack) 1!2(req) 1?2(ack)

1!2(req) 2?1(req) 2?1(req)

And following is the illustration of the flow in which this automaton changes the states:

Process 1	Process 2	Channels
$s_0 \xrightarrow{1!2(req)} s_0$		req
$s_0 \xrightarrow{1!2(req)} s_0$		req,req
	$t_0 \xrightarrow{2?1(req)} t_1$	req
	$t_1 \xrightarrow{2!1(ack)} t_0$	req AND ack
	$t_0 \xrightarrow{2?1(req)} t_1$	EMPTY AND ack

	$t_1 \xrightarrow{2!1(ack)} t_2$	EMPTY AND ack,ack
$s_0 \xrightarrow{1?2(ack)} s_1$		EMPTY AND ack
$s_1 \xrightarrow{1!2(req)} s_2$		req AND ack
$s_2 \xrightarrow{1?2(ack)} s_1$		req AND EMPTY
$s_1 \xrightarrow{1!2(req)} s_2$		req, req AND EMPTY
	$t_2 \xrightarrow{2?1(req)} t_2$	req AND EMPTY
	$t_2 \xrightarrow{2?1(req)} t_2$	Both Channels Empty

4 Configurations and Linearizations of MPAs

4.1 Configuration of MPA

Let A be a MPA over P and C (as defined in the above sections) then a configuration "Conf$_A$" of MPA A can be formally defined as....

$$\text{Conf}_A := \text{S}_A \times \{\eta \mid \eta : \text{Ch} \rightarrow (\text{C} \times \text{D})^*\}$$

So a configuration of an MPA A defines the set of mappings between S_A (Set of all states in MPA for all the processes) and C (Set of channels). Similarly, the global step for the MPA A can be defined as..

$$\Rightarrow \text{A} \subseteq \text{Conf}_A \times \text{Act} \times \text{D} \times \text{Conf}_A$$

sending a message:(($\bar{s}$, η), p!q(a) , m , ($\bar{s}'$, η')) $\subseteq \Rightarrow$ A if

- ($\bar{s}$[p] , p!q(a) , m , $\bar{s}'$[p]) $\subseteq \triangle_p$
- $\eta' = \eta$[(p, q)/(a,m) . η((p, q))]
- $\bar{s}$[r] = $\bar{s}'$[r] for all r $\subseteq$ P and r $\neq$ p

receiving a message:(($\bar{s}$, η), p?q(a) , m , ($\bar{s}'$, η')) $\subseteq \Rightarrow$ A if

- ($\bar{s}$[p] , p?q(a) , m , $\bar{s}'$[p]) $\subseteq \triangle_p$
- $\eta = \eta'$[(q , p)/η'(q , p) . (a , m)]

- $\bar{s}$[r] = $\bar{s}'$[r] for all r $\subseteq$ P and p$\neq$q

4.2 Run of MPA

A run ρ of an MPA A on $\sigma_1 \ldots \sigma_n \in$ Act* is a sequence

$$\rho = \gamma_0 \mathrm{m}_1 \gamma_1 \ldots \gamma_{n-1}\ \mathrm{m}_n \gamma_n$$

such that

γ_0 = (s_{init}, η_ε) with η_ε mapping any channel to ε

$\gamma_{i-1} \xrightarrow{s_i, m_i}$ A γ_i for any i $\in$ 1, . . . , n

Run ρ is an accepting run, if $\gamma_n \in$ F $\times$ η_ε , that is if for any run we get to the any of the accepting states of MPA then that run is an "accepting run".

4.3 Linearizations of MPA

The set of linearizations Lin(A) of an MPA A can be defined as "The set of all sequences of actions for each of which there exists an accepting run of A". Formally...

$$\mathrm{Lin(A)} = \{\ \mathrm{w} \in \mathrm{Act}^* \mid \text{there is an accepting run of A on w}\ \}$$

5 Undecidability of nonemptiness in MPAs

For any MPA A over P and C, it is undecidable whether the set of MSCs with in MPA (L(A)) of A would be an empty set or not (even if C is a singleton). Formally...

$$\mathrm{L(A)} = \emptyset?$$

The proof of the above problem is being provided by the reduction from the Turing Machine's halting problem. Since it is undecidable whether Turing machine(TM) will halt or not, the existance of this mapping yields that the emptiness problem for A is undecidable. We built an MPA A = ((A_1,A_2), D , s_{init} , F) over P = {1, 2} and some singleton set C such that

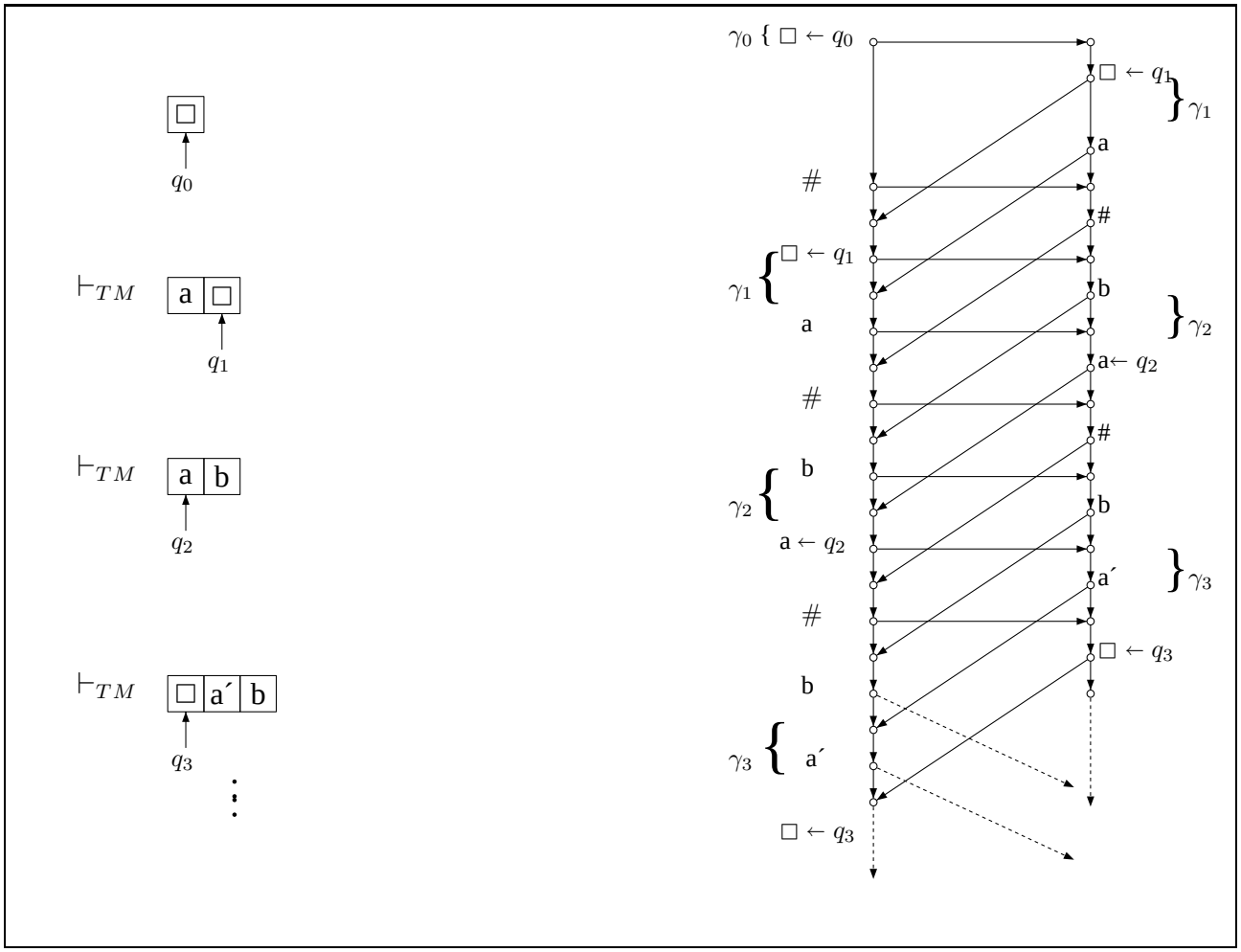
$$\mathrm{L(A)} = \emptyset \text{ iff TM can reach } q_f$$

5.1 Proof (sketch):

Reduction from halting problem for Turing machine TM = (Q , $\sum$, $\triangle$, $\square$, q_0, q_f)
to emptiness for MPA with two processes. Build MPA A = ((A_1,A_2),D, s_{init} , F) over {1, 2}
and some singleton set such that L(A) $\neq \emptyset$ iff TM can reach q_f.

- Process 1 sends current configurations to process 2.
- Process 2 chooses successor configurations and sends them back to 1.
- D = (($\sum \cup \{\square\}$) $\times$ (Q $\cup$ {-})) $\cup$ {#}

The figure on the next page shows the steps of execution of such a TM...



• Left or standstill transition:

Process 2 may just wait for a symbol containing a state of the Turing machine and to alter it correspondingly. In the example, the left-moving transition (q2 , a , a´ , L , q3) is applied so that process 2

- sends b unchanged back to process 1
- detects (receives) a ← q_2
- sends a´ to process 1 entering a state indicating that the symbol to be sent next has to be equipped with q_3

• receives # so that the symbol □ ← q3 has to be inserted before returning #

• Right transition:

Process 2 has to guess what the position right before the head is. For example, provided process 2 decided in favor of (q_2, a, a´, R, q3) while reading the b, it would have to

• send b ← q_3 instead of just b, entering some state t (a ← q2)

• receive a ← q_2 (no other symbol can be received in state t (a ← q2))

• send a´ back to process 1

• Introduce local final states s_f and t_f , one for process 1 and one for process 2, respectively (i.e., F = (s_f , t_f) and A is locally accepting).

• At any time, process 1 may switch into s_f , in which arbitrary and arbitrarily many messages can be received to empty channel (2, 1).

• Process 2 is allowed to move into t_f and to empty the channel (1, 2) as soon as it receives a letter c ← q_f for some c.

• As process 2 modifies a configuration of TM locally, finitely many States are sufficient in A.

6 Sub classes of MPAs: Boundedness

6.1 Definition of B-Bounded Words

A word w ∈ Act* is called B-Bounded (where natural number B ≥ 1), if for any u ∈ Pref(w) and any channel (p,q) ∈ Ch:

$$\sum_{a \in C} |u|_{p!q(a)} - \sum_{a \in C} |u|_{p?q(a)} \leq B$$

That is, in w, it is never the case that the number of sends on channel (p,q) is more than B ahead of the number of receives on (p,q) by process q.

Conversely, we can also say that for any given MPA A, the buffers with maximum size B (for all the channels) would be sufficient to execute a word w ∈ Act*. If this condition holds, then word w will be called B-Bounded.

6.2 Universal Boundedness of MSCs

An MSC M is called universally B-bounded if all the linearizations of M are B-Bounded. Formally...

$$\text{Lin(M)} = \text{Lin}_{\text{B}}\text{(M)}$$

Where

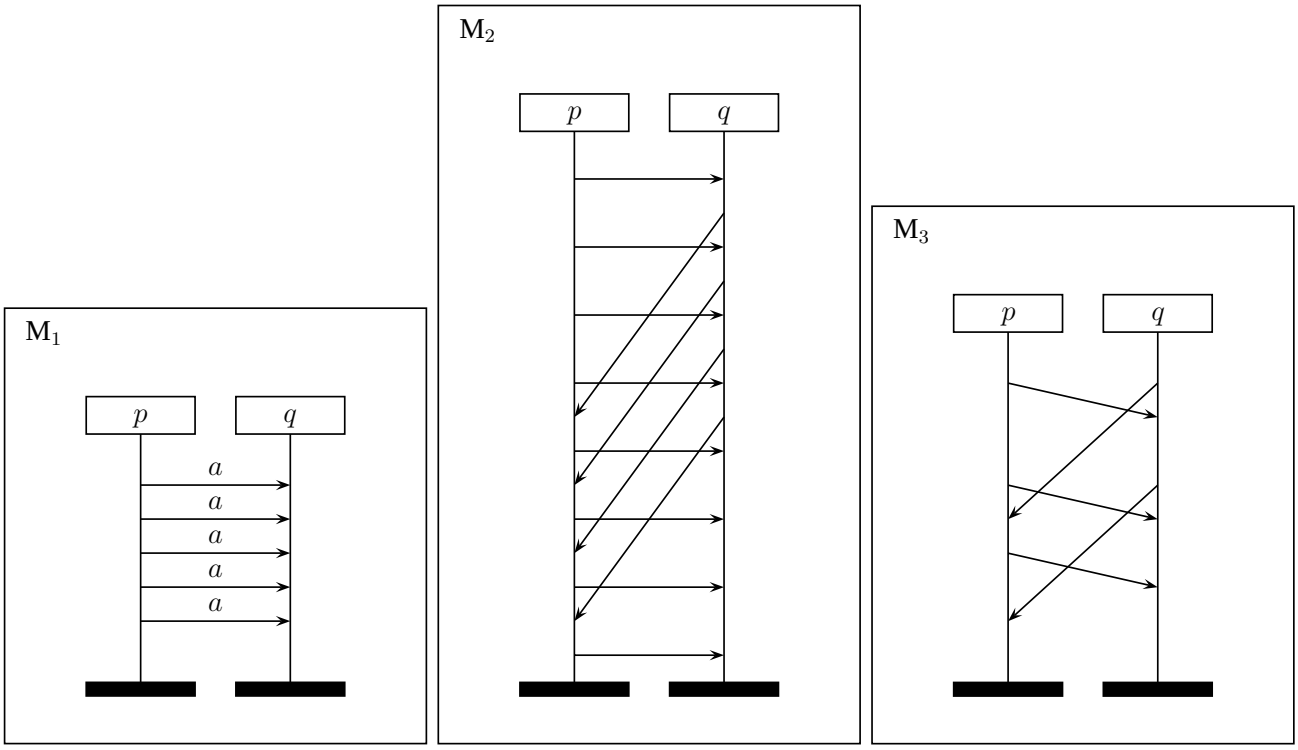
- Lin(M) = set of all linearizations of M
- Lin_{B}(M) = set of all B-Bounded Linearizations of M = {w $\in$ Lin(M) | w is B-Bounded}

6.3 Existential Boundedness of MSC

An MSC M is called existentially B-Bounded if there exists some linearization of M which is B-Bounded. Formally...

$$\text{Lin(M)} \bigcap \text{Lin}_{\text{B}}\text{(M)} \neq \emptyset$$

To check boundedness property of a system, all the channel buffers must be checked for verification. Let's look at the following MSCs as an example:



The MSC M_1 is

- Universally 5-Bounded (All linearizations of M_1 are 5-Bounded)
- Existentially 1-Bounded (some linearizations of M_1 are 1-Bounded)

The MSC M_2 is

- Universally 4-Bounded (All linearizations of M_2 are 3-Bounded)
- Existentially 2-Bounded (some linearizations of M_2 are 1-Bounded)
- But not existentially 1-Bounded (no linearization of M_2 are 2-Bounded)

The MSC M_3 is

- Universally 3-Bounded (All linearizations of M_3 are 5-Bounded)
- Existentially 1-Bounded (some linearizations of M_3 are 1-Bounded)

——————END——————