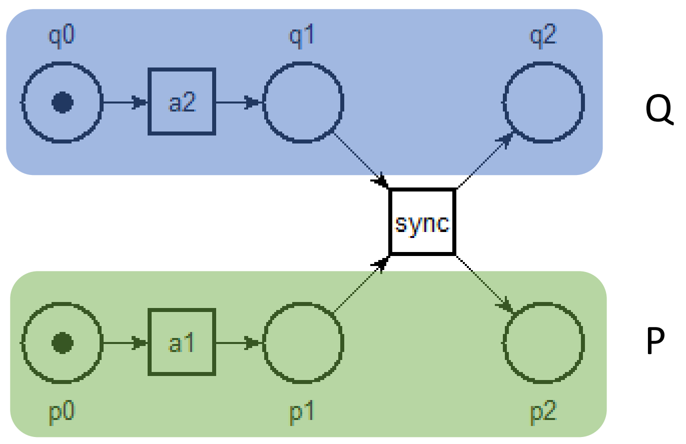
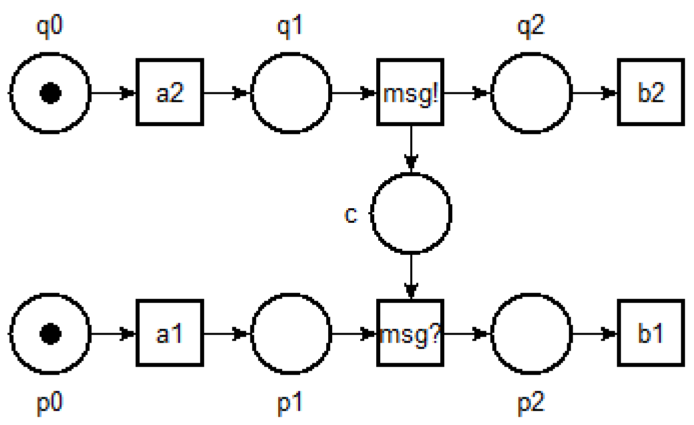
Introduction to Model-Checking
Part 3— Resource Allocation

Background. Petri nets transitions provide a direct way to encode communication between processes by synchronous communication. The following example shows two “processes” (P and Q) that can make an internal computation then synchronize with each other before continuing.



We can model asynchronous communication just as well using “communication places”. For instance, in the following net, process Q can send a message *msg* (*msg!* is for emission and *msg?* stands for reception) and then continue with an internal action b_2 before process P can receive it. Often, we need to add acknowledgment messages to ensure that messages are not queued up (place *c* is bounded).



We will use this modeling approach in the following exercises.

Exercise 1. Resource allocation between one client and two managers

A client (C1) needs to access two resources (R1 and R2) in mutual exclusion in order to work. Thin of the example of the Printer job queue example. The resources are managed remotely by two different resource managers (M1 and M2).

Communication between the client and the managers relies on three different messages (signals)

- **Req** request to access the resource
- **Ack** acknowledgment authorizing the use of the resource
- **Free** client has finished and signal that the resource is freed

The behaviors of C1, M1 and M2 is as follows.

Client C1 asks for the two resources sequentially; one after the other. First he asks for R1 then, when it as the resource, he asks access to R2. When he has both resources he enters into critical section (he works). Afterwards he frees the two resources simultaneously and comes back to its initial state.

Manager Gi ($i \in \{1,2\}$) initially awaits for a reception of message Req from a client. Upon reception, he allows exclusive access to the resource to the client with message Ack. If the resource is not available, the request sits idle. The manager knows when the resource is available again when he receives message Free.

1. Model the system with a P/T net (see remark below)
2. Simulate the global (composed) system using the stepper
3. Build the marking graph and check whether there is a deadlock
4. Check whether the system is reinitialisable (check the SCC)

Remark: to build the model, you can start from the description of the client C1 below. This is a Petri net in textual form (.net format).

```
# file cl1.net
## places Cl1_M1_req, Cl1_M1_release and M1_Cl1_ack
## corresponds to messages sent between C1 and M1.
## We use a naming convention Sender_Destination_Message

pl Cl1_idle (1)

tr Cl1_req1 Cl1_idle -> Cl1_wait1 Cl1_M1_req
tr Cl1_req2 Cl1_wait1 M1_Cl1_ack -> Cl1_wait2 Cl1_M2_req
tr Cl1_enter Cl1_wait2 M2_Cl1_ack -> Cl1_work
tr Cl1_exit Cl1_work -> Cl1_M1_free Cl1_M2_free Cl1_idle
```

You can use nd to draw the net. Then, taking inspiration from the client, write two separate net files for M1 and M2 (say in file m1.net and m2.net). Finally, compose the three files to build the whole system. To compose your sub-nets you can use the tpn format, e.g. use a script file (say sys.tpn) and open it with nd. In case of compilation error, you can use the command **tina –p sys.tpn**.

```
# file sys.tpn
# you can mix textual (.net) and graphical (.ndr) files
source cl1.net
source m1.ndr
source m2.net
```

Exercise 2. Resource allocation between two clients and two managers

We consider a second client (C2) that has a behavior equivalent to C1 except that he asks for resource R2 before R1.

1. Model the second client and add it to the previous system using place composition as before. You will need to change the description of the two managers.
2. Using the stepper, find two scenarios that show the possibility of:
    a. Deadlock
    b. Starvation (a client that wants to work will be forever denied access to the resources)
3. We want to check mutual exclusion between the two clients working; meaning that places Cl1_work and Cl2_work cannot both have a token at the same time.
    a. How can you show the property on the marking graph
    b. Add a transition to your system that can fire if both places are marked and show that it is dead
4. Now check mutual exclusion on the use of each one of the resources.