

# Process and Data Modeling

Extended Finite Automata, Timed Automata, and Model Checking

— A Formal Approach to Modeling

Jens Chr. Godskesen  
IT University of Copenhagen

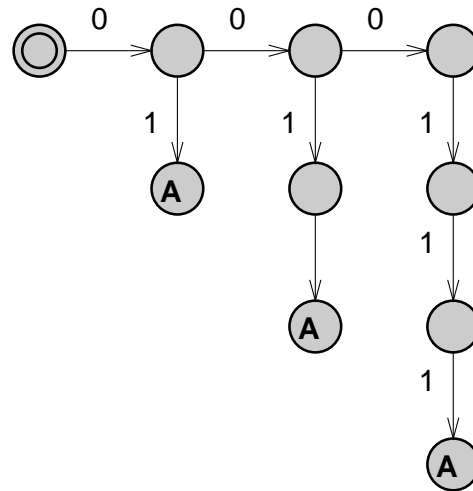
# Overview

- A walk through selected exercises from last lecture
- Extended Finite Automata
  - Variables and guards
  - Templates and parameters
- Timed (Extended) Finite Automata

## Exercises previous lecture (I)

?(exercise 4 slide 15): Define a language that is non-regular.

$L = \{0^n 1^n \mid n > 1\}$  isn't regular, because the states of the first part of a hypothetical FA accepting  $L$  would look like:



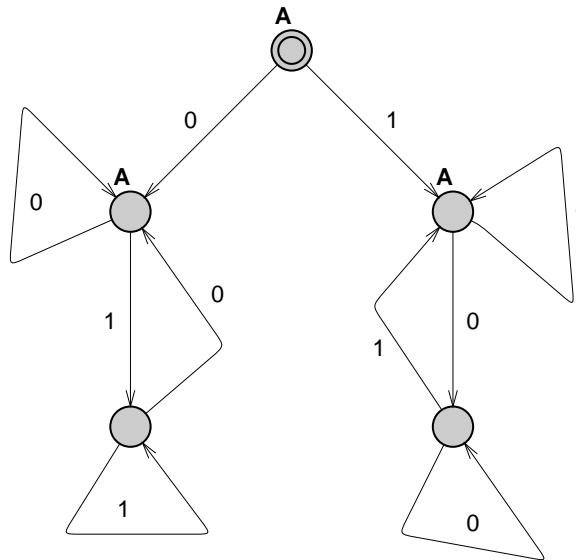
Any such FA accepts only strings of a fixed maximal length.

## Exercises previous lecture (II)

?(exercise 5 slide 15): Is the language  $L$  regular?

$L = \{\omega \in \{0, 1\}^* \mid \omega \text{ has an equal number of } 01 \text{ and } 10 \text{ as substrings}\}$

Observe that  $101 \in L$ , but  $1010 \notin L$



## Exercises previous lecture (III)

?(exercises 4, 5, and 7 slide 28):

- Define two CFA's, one being a vending machine selling coffee and one being a customer buying coffee. Let the coffee machine require 7 Dkr. pr. cup of coffee. The input to the coffee machine should be coins of Dkr. 1, 2, and 5. By convenience, let the user always be willing to insert any coin and receive coffee. Try to understand your model by simulation.
- Define relevant queries to your system from exercise 4 and check the queries in Uppall. For instance, it should be possible for the user to get a cup of coffee. Does Uppall find the the same trace depending on whether you choose to find a shortest diagnostic trace or just some trace?
- Let your system from exercise 4 consists of two users, make sure that buying coffee is carried out as a critical section. How to validate the last property? Does your queries from before still hold.

# Extended Finite Automata (I)

A CFA may be extended with **variables**. The execution of a transition may depend on the value of variables.

$$p \xrightarrow{a!, x > 10} p'$$

means that there is a transition from  $p$  to  $p'$  but only if the value of  $x$  is greater than 10. A *boolean condition*, like  $x > 10$ , is called a **guard**.

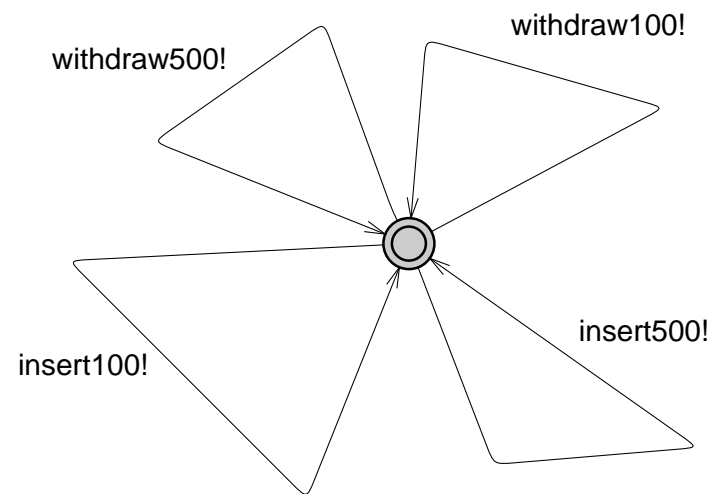
The values of variables may be **assigned** when performing transitions.

$$p \xrightarrow{a?, x > 10, y := 5} p'$$

sets the variable  $y$  to 5.

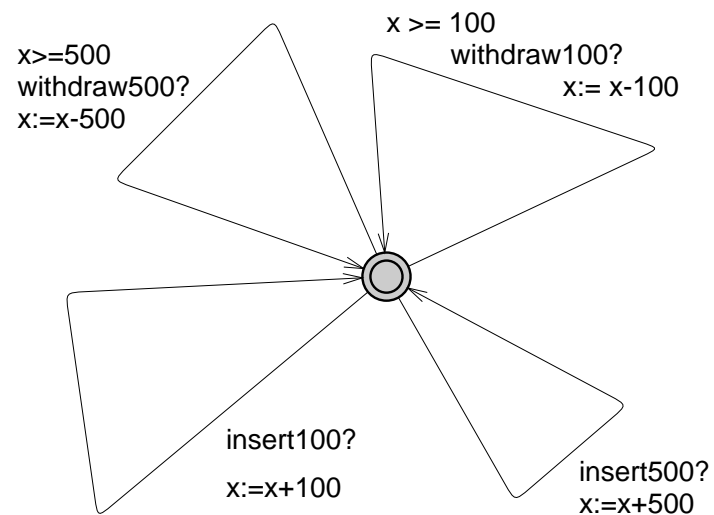
# Extended Finite Automata (II)

Here is a *user* of a bank account:



# Extended Finite Automata (III)

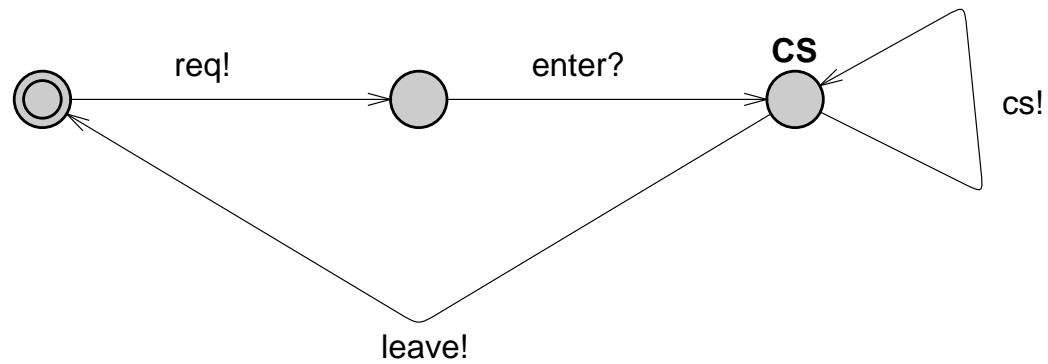
and the *account* is specified by:



The account must never be negative, let's check  $A[] \quad x \geq 0$ .

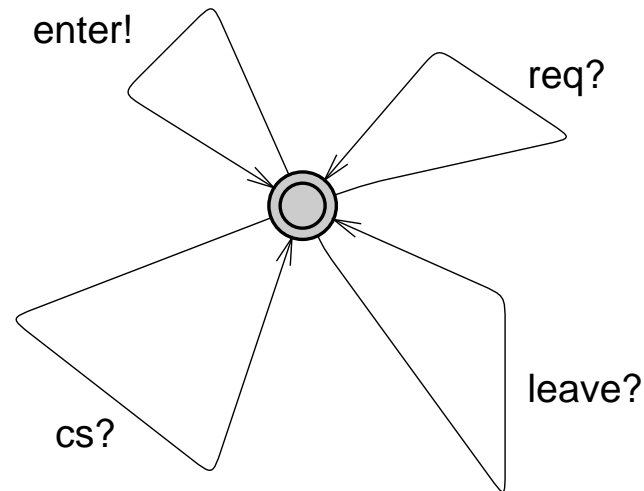
# Extended Finite Automata (IV)

Suppose a user requesting for entering a *critical section*. A template for such a user could be specified by:



# Extended Finite Automata (V)

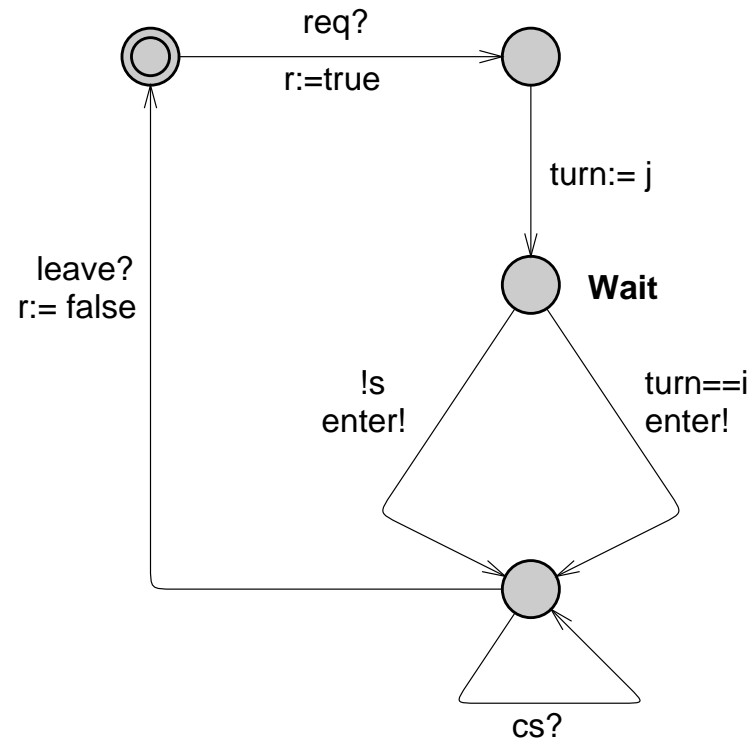
We let the controller (template) for the user be defined by:



Instantiating two users, P1 and P2, with each their controller we observe that  $A[]$   $(P1.CS \text{ imply } !P2.CS)$  and  $(P2.CS \text{ imply } !P1.CS)$  doesn't hold.

# Extended Finite Automata (VI)

To obtain mutual exclusion the controller may be replaced by Pettersons algorithm (see slide 4 from previous lecture):



## Extended Finite Automata (VII)

Now, two users, P1 and P2, each controlled by Peterson algorithm may never enter their critical section simultaneously.

Let's study the model and let's check

$A[] (P1.CS \text{ imply } !P2.CS) \text{ and } (P2.CS \text{ imply } !P1.CS)$

# Uppall Exercises

1. Modify your model of a borrower such that you keep track of how many books can be borrowed and such that there is an upper limit as to how many books a borrower can borrow. Check properties of your model.
2. Redo the coffee vending machine from the exercises from the previous lecture, but now using variables in order to minimize the number of states.
3. Extend your machines such that also coins of Dkr. 10 can be given as inputs to the coffee machine and let it pay back the surplus.
4. Check that the properties from your previous coffee machine at the last lecture does still hold, and ask perhaps auxiliary queries.
5. Change Pettersons algorithm such that two processes controlled by it may enter their respective critical sections simultaneously. Check you changed version in Uppall and explain your answer.

# Timed Finite Automata (I)

A **timed CFA** is an extended CFA containing also *real valued clocks*.

A clock measures the progress of time and may be part of a guard. For instance we may have a transition:

$$p \xrightarrow{a!, c > 10} p'$$

meaning that there is a transition from  $p$  to  $p'$  but only if the time of clock  $c$  is greater than 10.

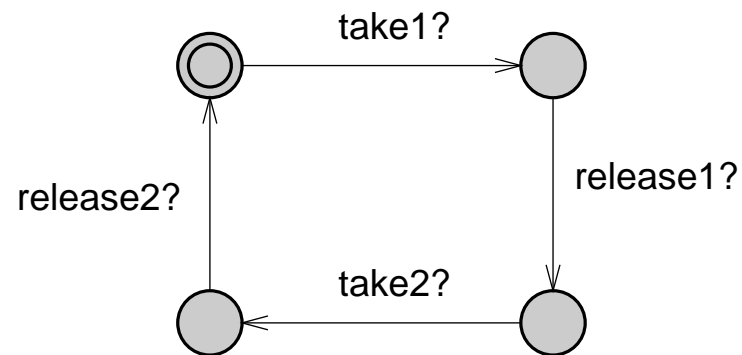
Clock values may be set when performing transitions, so

$$p \xrightarrow{a!, c > 10, c := 0} p'$$

also *resets* the clock  $c$ .

## Timed Finite Automata (II)

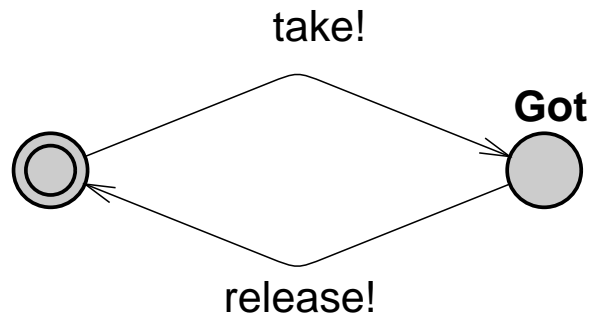
Consider a variant of the riddle with a torch



and let the system have a **global** clock  $t$  with initial value 0.

## Timed Finite Automata (III)

Put the torch together with two soldiers *Soldier1* and *Soldier2* based on the template



such that *Soldier1* has channels *take1* and *release1*, and *Soldier2* has channels *take2* and *release2*.

## Timed Finite Automata (IV)

When the system is started time begins to progress and the value of  $t$  increases.

*Soldier1* is apparently infinitely fast since the following property holds:

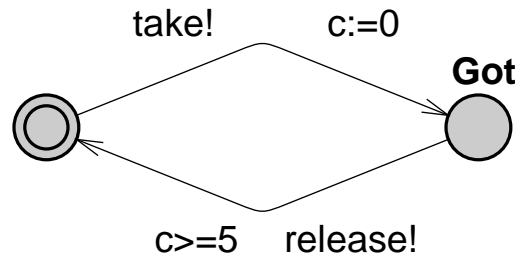
$E \langle \rangle (\text{Soldier2.Got and } t == 0)$

i.e. he takes and releases the torch before time progresses. But time may also progress since

$E \langle \rangle (\text{Soldier2.Got and } t == 5)$

# Timed Finite Automata (V)

To be able to more precisely model duration of events we add a **local** clock  $c$  to a soldier and define



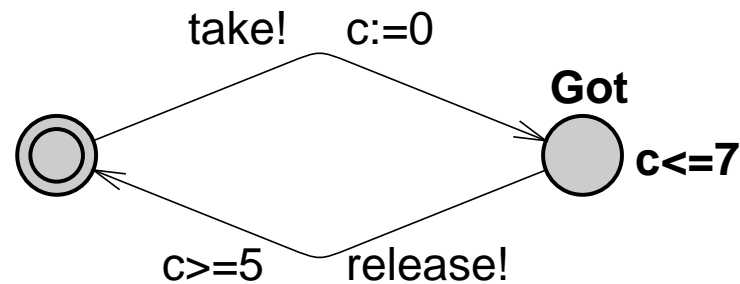
meaning that the torch must be kept for at least 5 time units before released.

Now  $E \langle \rangle \text{Soldier2.Got}$  and  $t < 5$  is false, but since *Soldier1* may keep the torch as long as he wants, it holds that

$A[] \text{Soldier2.Got} \text{ imply } t \geq 5$

## Timed Finite Automata (VI)

To restrict the amount of time allowed to stay in a state we may enforce *time progress* using **invariants**. We may e.g. require that a soldier doesn't keep the torch for more than 7 time units:

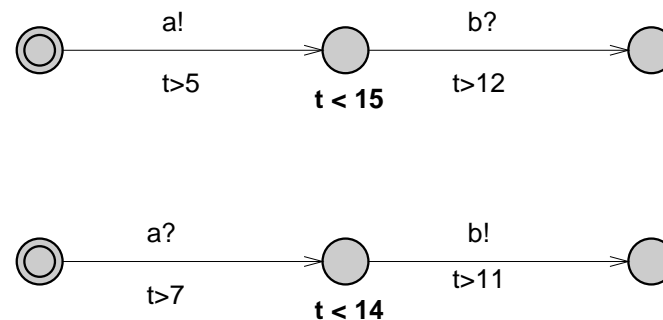


Now, it holds that:

$A[] \text{ Soldier2.Got imply Soldier2.c } \leq 7$

## Timed Finite Automata (VII)

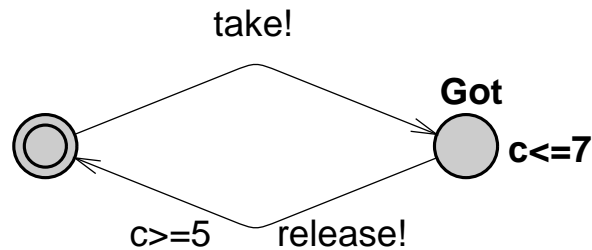
A system is **deadlocked** if no process can reach a new state by doing an input or output action, even by letting time progress. For instance, the system



may deadlock either after doing  $a$  and  $b$ , or because it has been waiting more than 14 time units in its initial state.

## Timed Finite Automata (VIII)

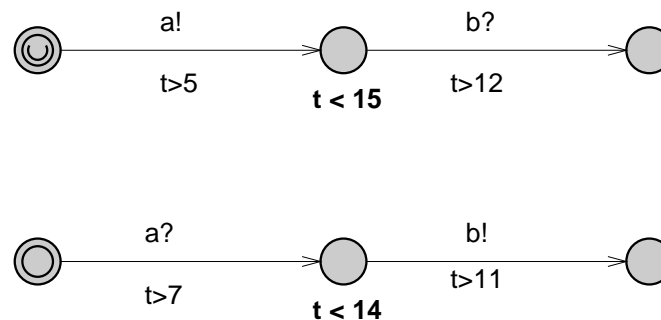
As another example, a deadlock may occur when leaving out the reset of  $c$  in the model of a soldier:



It makes the system satisfy  $E \langle \rangle \text{ deadlock}$ , because a soldier may stay in its initial state more than 7 time units.

## Timed Finite Automata (VIII)

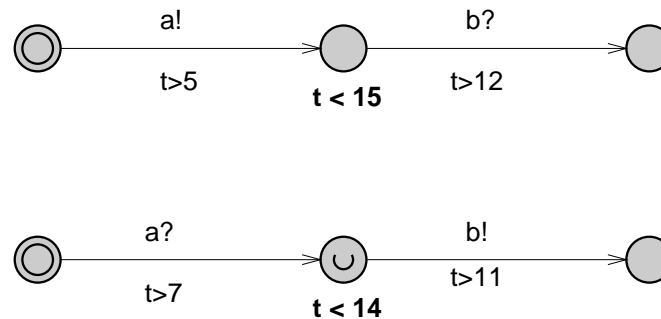
A state may be declared as **urgent** (denoted by  $U$ ) meaning that it is not allowed to stay in the state, so it must be left immediately.



Above the system deadlocks because the initial state must be left at once, but that cannot happen due to the time constraints.

# Timed Finite Automata (VIII)

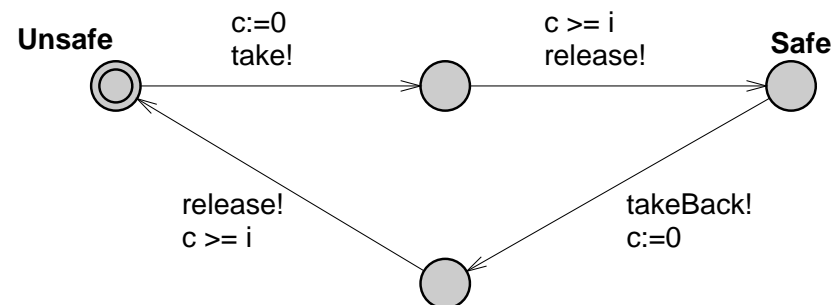
Changing the urgent state such that



imply that the system may deadlock in its initial state if it waits at least 14 time units. If it waits between 7 and 12 time units it may do an  $a$  after which it deadlocks, if it waits between 12 and 14 time units it may do an  $a$  and immediately afterwards a  $b$ .

# The Riddle Revisited (I)

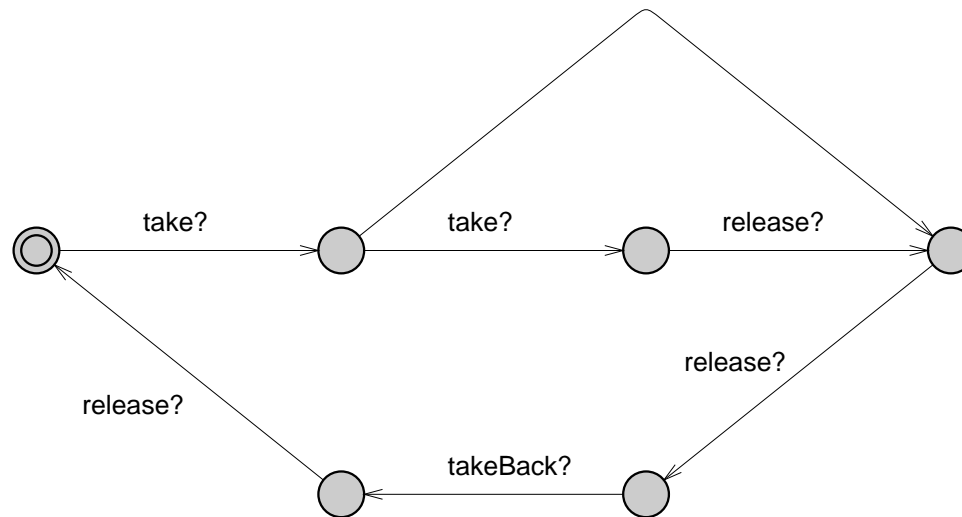
In the model of our riddle a soldier with clock  $c$  is defined by



where a torch *take*-transition resets the clock  $c$  and a *release* transition is first enabled  $i$  time units after the torch was taken (the time it takes the soldier to cross the bridge).

## The Riddle Revisited (II)

... and the (timed) version of the torch is defined by

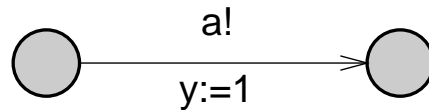


One or two soldiers may take the torch and cross the bridge to the safe side. Only one soldier can bring the torch back again.

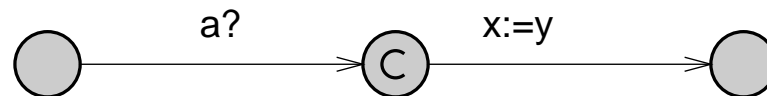
# Value Passing (I)

Value passing may be encoded using **committed states**, i.e. a state where an outgoing transition must be taken immediately.

E.g. output may be encoded by:

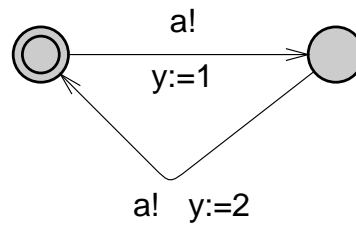


and input by:

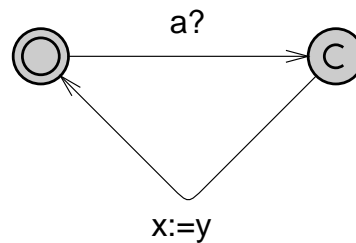


# Value Passing (II)

Simulate the system consisting of one output process and two input processes defined as below.

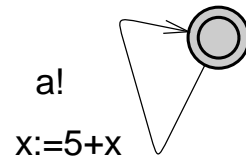


and input by:

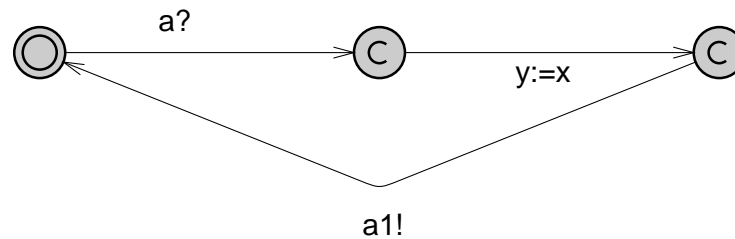


# Broad Cast (I)

Broad cast may be encoded by a series of committed steps. Suppose an output process;

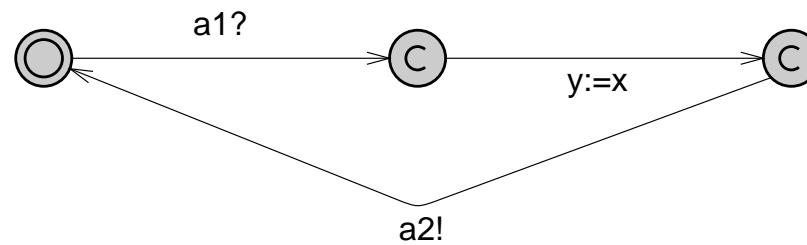


The first receiver may be defined by:

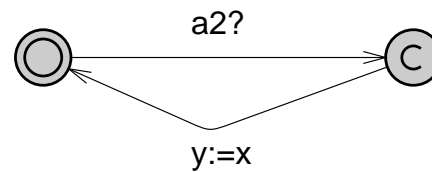


## Broad Cast (II)

The next receiver may be defined by:



and the final receiver by:



# Exercises

1. Go through the exercises at slide 13 we didn't make during the lecture
2. Which states in the torch defined on slide 25 can be made urgent without the whole system deadlocks?
3. Why isn't encoding of value passing by two synchronizing transitions,  $p \xrightarrow{a!, y := 3} p'$  and  $q \xrightarrow{a?, x := y} q'$  say, valid? Give a model that illustrates your argument.
4. Redefine the modified version of your library (exercise 1 slide 13) such that time is involved. As an example you may model

that a book can only be borrowed for a certain amount of time, and also you may model that it takes at least some time to read a book.

5. Adapt your model of the library from above such that you may ask how long it takes the borrowers to read the books at least once.