Prof. Bernd Finkbeiner
Dr. Calogero Zarba
Moritz Hahn

Sample Solution for Problem Set 1
Due: April 24, 2007, 9:14 am

**Embedded Systems**

---

# Problem 1

(a)

| | |
|---|---|
| $A_1.0 \xrightarrow{\text{coin\_in/ok}} A_1.1$ | (insertion of coin by user) |
| $A_2.A \xrightarrow{\text{ok/}} A_2.B$ | (caused by event 'ok') |
| $A_2.B \xrightarrow{\text{req\_tea/start\_tea}} A_2.D$ | (selection by user) |
| $A_2.D \xrightarrow{\text{drink\_ready/done}} A_2.A$ | (caused by environment) |
| $A_1.1 \xrightarrow{\text{done/}} A_1.0$ | (caused by event 'done') |

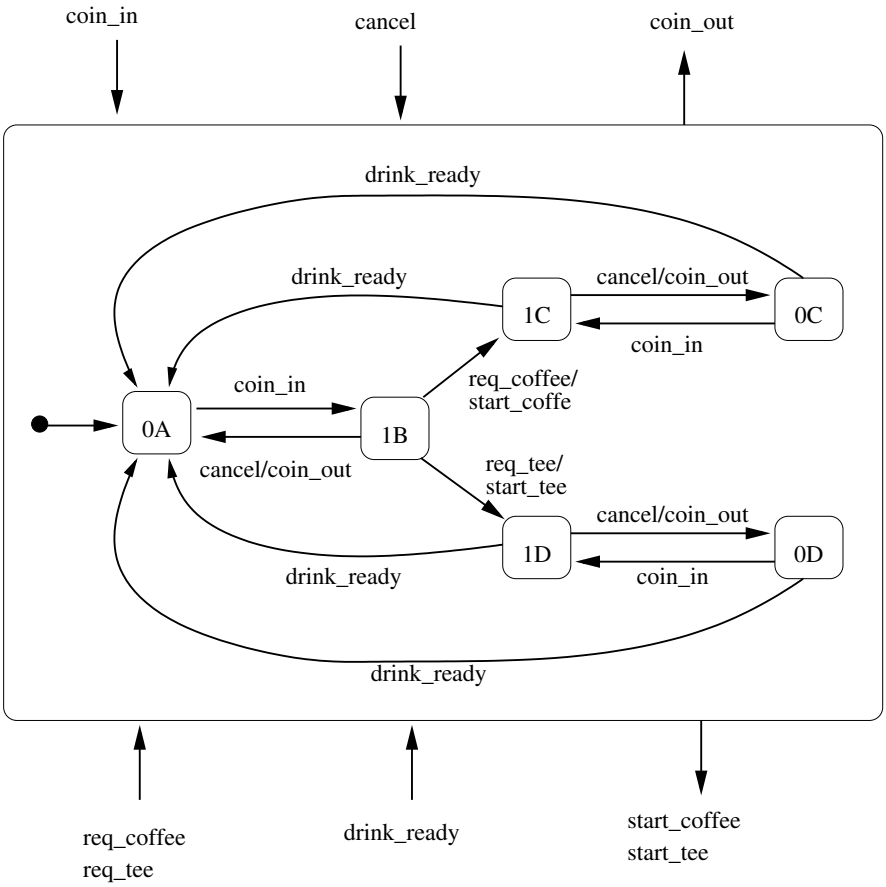
(b) The following trace of transitions describes the scenario

| | |
|---|---|
| $A_1.0 \xrightarrow{\text{coin\_in/ok}} A_1.1$ | (insertion of coin by user) |
| $A_2.A \xrightarrow{\text{ok/}} A_2.B$ | (caused by event 'ok') |
| $A_2.B \xrightarrow{\text{req\_tea/start\_tea}} A_2.D$ | (selection by user) |
| $A_1.1 \xrightarrow{\text{cancel/coin\_out,reset}} A_1.0$ | (user presses reset key) |
| $A_2.D \xrightarrow{\text{drink\_ready/done}} A_2.A$ | (caused by environment) |

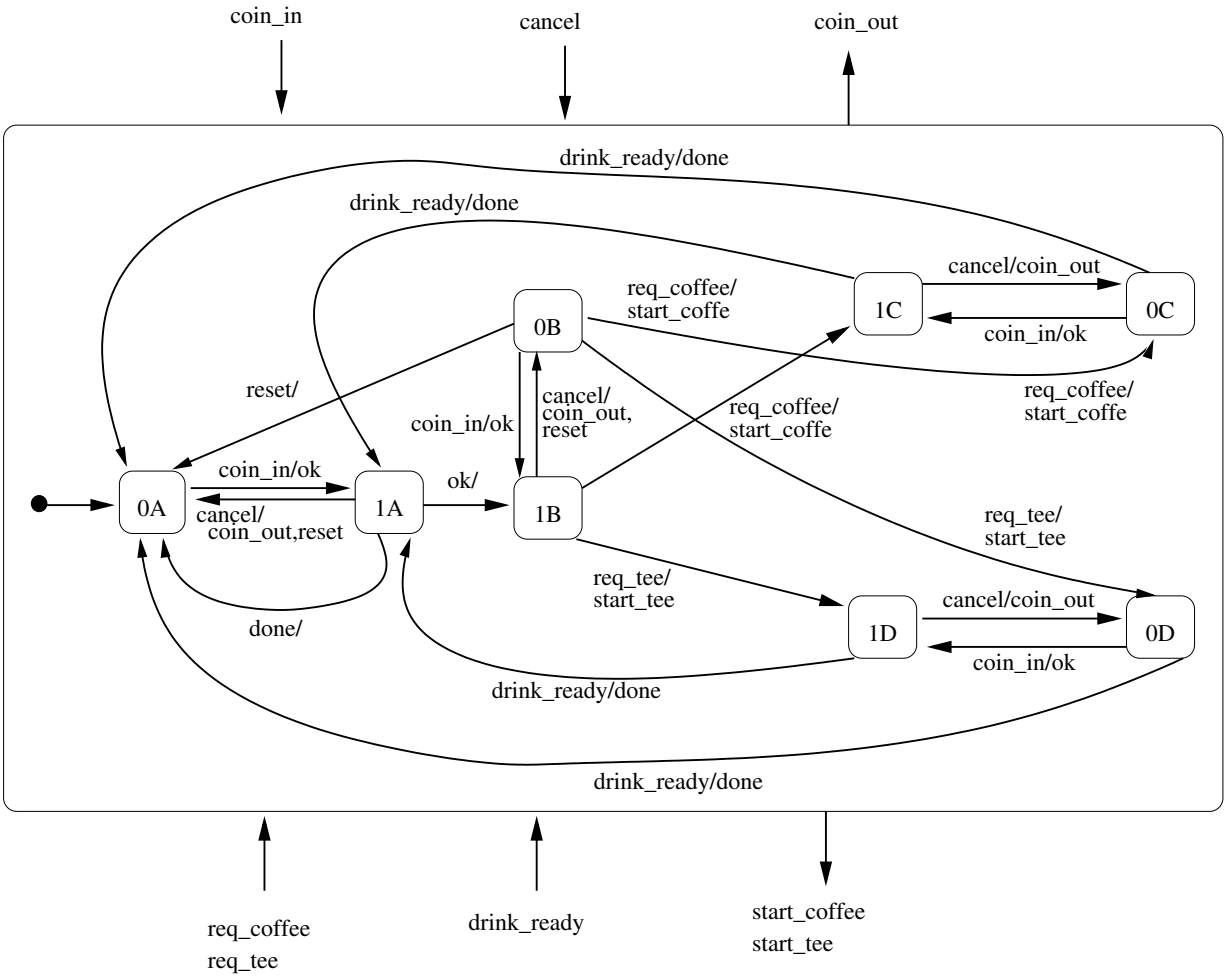
Here, the user orders a drink but presses the reset key while the drink is being prepared. This will cause the inserted coin to be thrown out, but the drink will be handed to the user anyway.

(c)

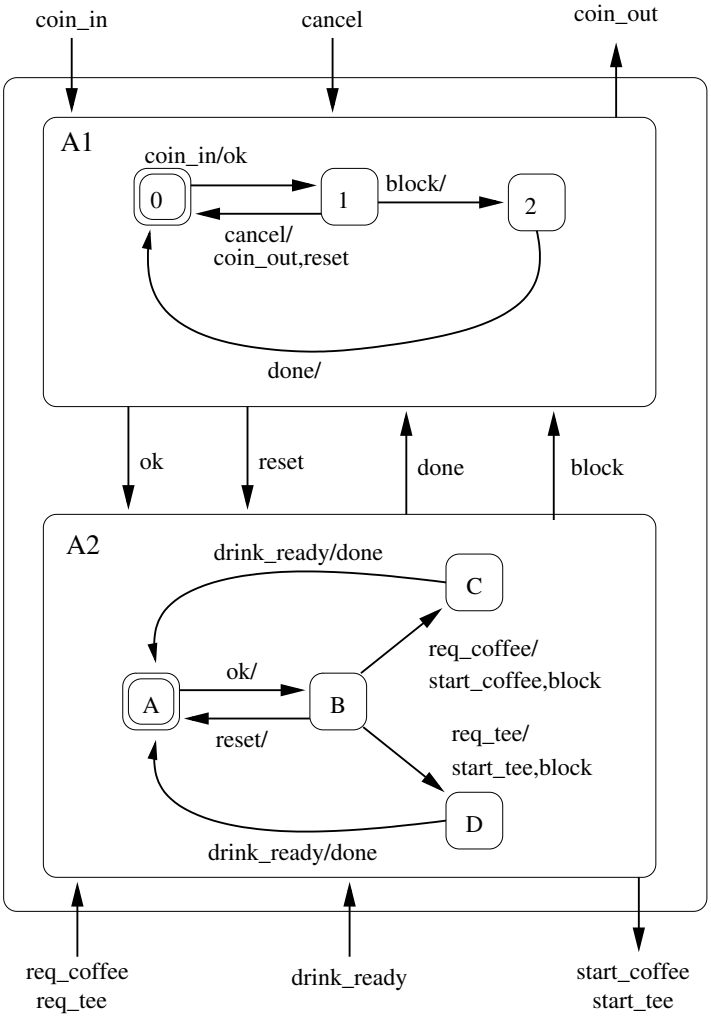
For the asynchronous semantic:




Note that the internal actions have been left out. If an internal action in the one parallel part of the automaton caused an immediate change in the other part, this change is represented by just one arrow here.
For the synchronous semantic:

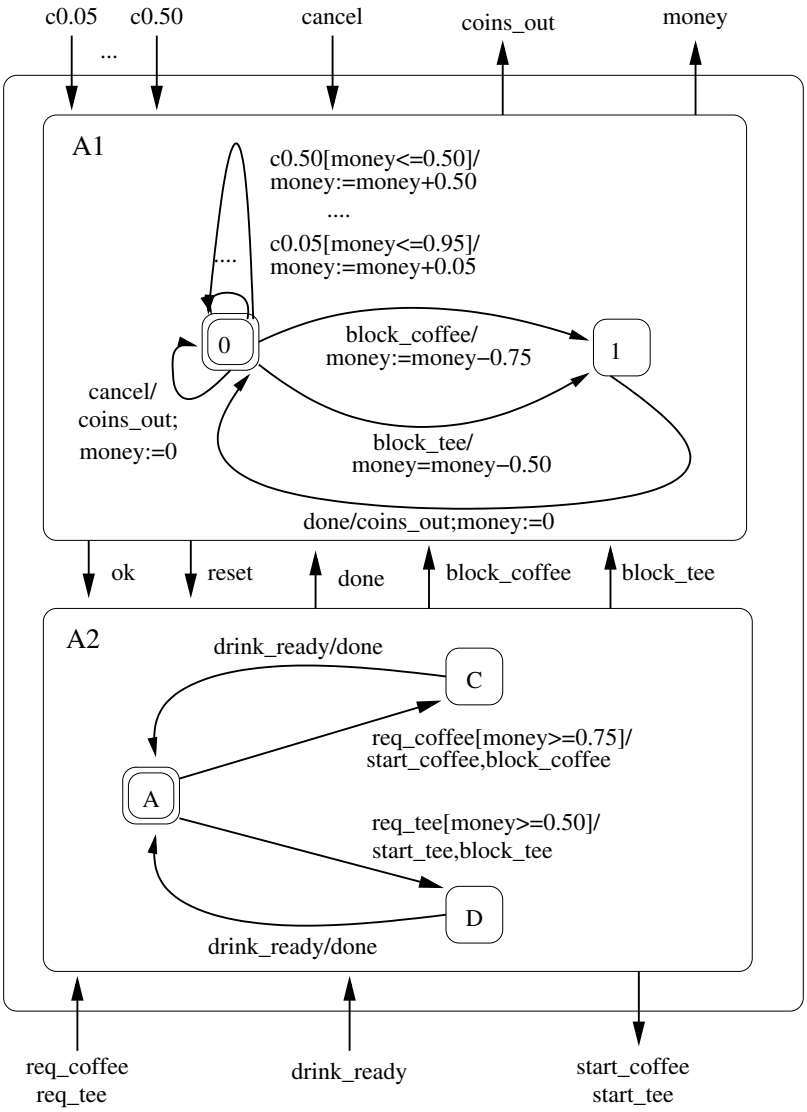



Here you also see the internal evens.
You may notice that the synchronous semantic makes not much sense for this model, as it allows the user to interrupt the internal calculations of the automaton.
(d)




In this modified version we have a 'blocking' state A1.2 where we prevent the user from causing a reset as soon as the preparation of the drink has been started. This transition to this state is controlled by the event 'block'.

(e)

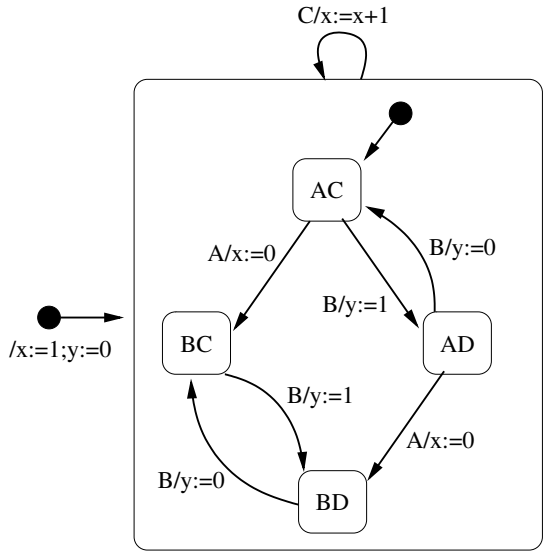



To allow the vending machine to accept different coins and have different prices for tea and coffee, we do the following modifications:

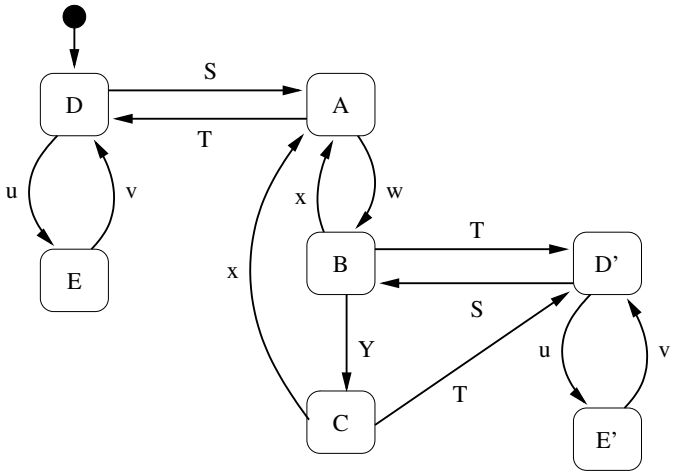
- we introduce a variable 'money' to remember the ammount of money the user has thrown into the automaton instead of having states to remember whether the user threw in a coin
- for each of the accepted coins, we introduce a new event 'cx' where x is the value of that coin
- we split the event 'block' to 'block_tea' and 'block_coffee' to allow the money management part of the automaton to handle the different prices for the two different drinks
- the event 'coin_out' has been substituted by the event 'coins_out'. As the event is sent to the environment, the automaton will output a number of coins of which the value is equal to 'money'. Therefore, we make 'money' an external output variable.

Notice that is is by far not the only correct solution, so your version may be quite different.

## Problem 2




## Problem 3




The primed versions of 'D' and 'C' indicate that we have been in the substate containing 'B' and 'C' before 'T' occured. Because the history connector is not a deep history connector and we don't have an additional history connector in the mentioned substate, you must not remember whether the last active state there was 'B' or 'C'!