Aufgabenblatt 4

4.1

Geben Sie für jeden der sieben im folgenden als Zustandsgraphen dargestellten Prozesse die zugehörige algebraische FSP-Beschreibung an. Überprüfen Sie Ihre Lösung mit Hilfe des Analysetools LTSA und geben Sie einige Traces aus.

a)

```
MEETING = (hello -> converse -> goodbye -> STOP).
// * hello -> converse -> goodbye
```

```
JOB = (arrive -> work -> leave -> JOB).
// * arrive -> work -> leave -> arrive -> ...
```

```
GAME = (one -> WIN
        |two -> WIN
        |three -> LOSE),
WIN = (win -> GAME),
LOSE = (lose -> GAME).

// * one -> win -> one
// * one -> win -> two
// * one -> win -> three -> lose -> one
// * two -> win -> three -> lose -> two
// * three -> lose -> one -> win -> three
// * one -> win -> one -> win -> one
// * three -> lose -> three -> lose -> three
```

```
DOUBLE = ( in[1] -> out[2] -> DOUBLE
         | in[2] -> out[4] -> DOUBLE
         | in[3] -> out[6] -> DOUBLE).
```

```
MOVE = (ahead -> AHEAD),
AHEAD = (left -> STOP | right -> MOVE).
// * ahead -> left
// * ahead -> right -> ahead -> left
// * ahead -> right -> ahaed -> right -> ahead -> left
// * ahead -> right -> ahead -> right ->
```

```
PERSON = WEEK,
WEEK = (weekday -> WEEKDAY|weekend -> WEEKEND),
WEEKDAY = (sleep -> work -> PERSON),
WEEKEND = (sleep -> (play -> PERSON|shop -> PERSON)).
// * weekday -> sleep -> work -> weekday -> sleep -> work
// * weekday -> sleep -> work -> weekend -> sleep -> play
// * weekend -> sleep -> shop -> weekday -> sleep -> work
// * weekend -> sleep -> play -> weekend -> sleep -> shop
// * weekday -> sleep -> work -> weekend -> sleep -> play -> weekday -> sleep -> work
```

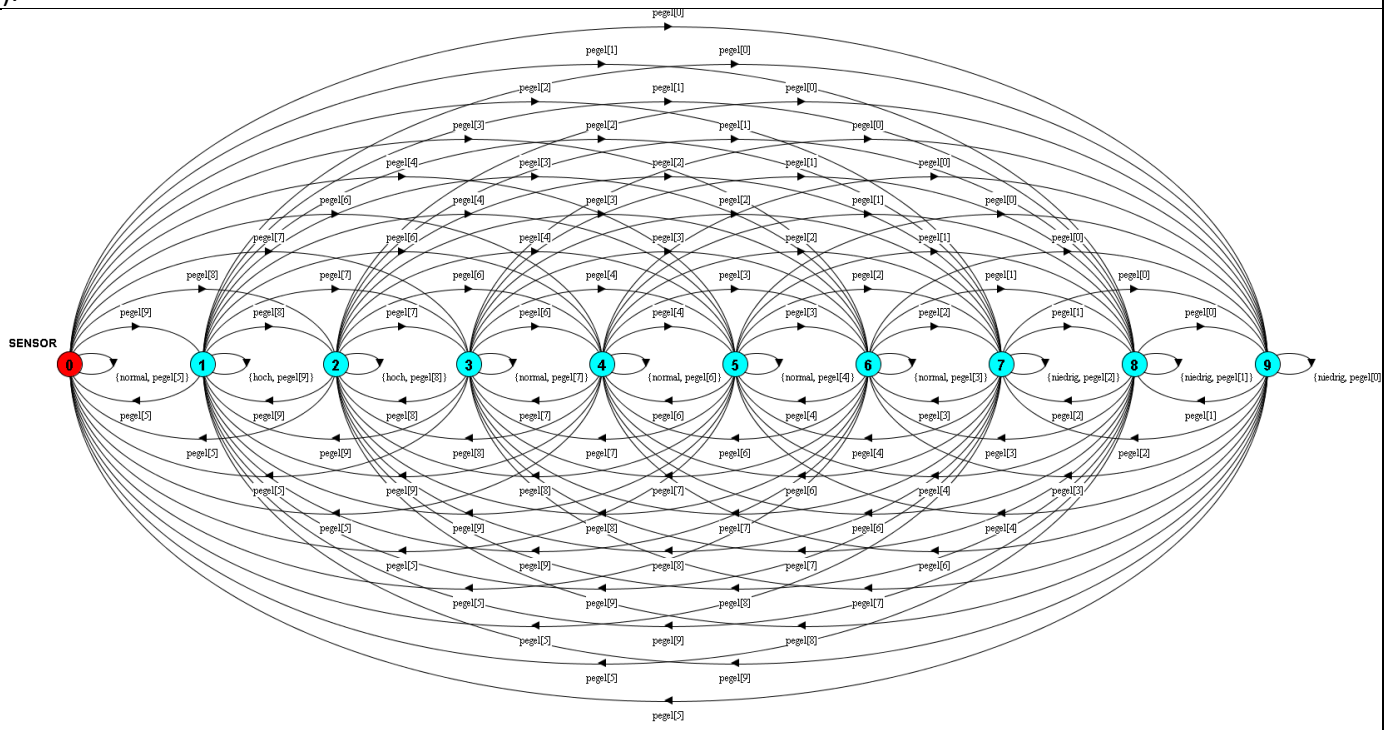
```
FOURTICK(N=4) = FOURTICK[0],
FOURTICK[i:0..N] = (when (i<N) tick -> FOURTICK[i+1]
                                    |when (i==N) STOP).
// * tick -> tick -> tick -> tick
// * tick -> tick -> tick
// * tick -> tick
// * tick
```

4.2

Ein Sensor misst den Wasserstand in einem Tank, wobei der Pegel die Werte $0, \ldots, 9$ annehmen kann (5 ist der Anfangspegel). In Abhängigkeit des Wasserstandes gibt der Sensor die Meldungen „zu niedrig“ (bei Pegel kleiner als 3), „zu hoch“ (bei Pegel größer als 7) oder „normal“ (sonst) aus. Modellieren Sie den Sensor als FSP SENSOR.

Hinweis: Das Alphabet von SENSOR ist $\{pegel[0..9], niedrig, hoch, normal\}$.

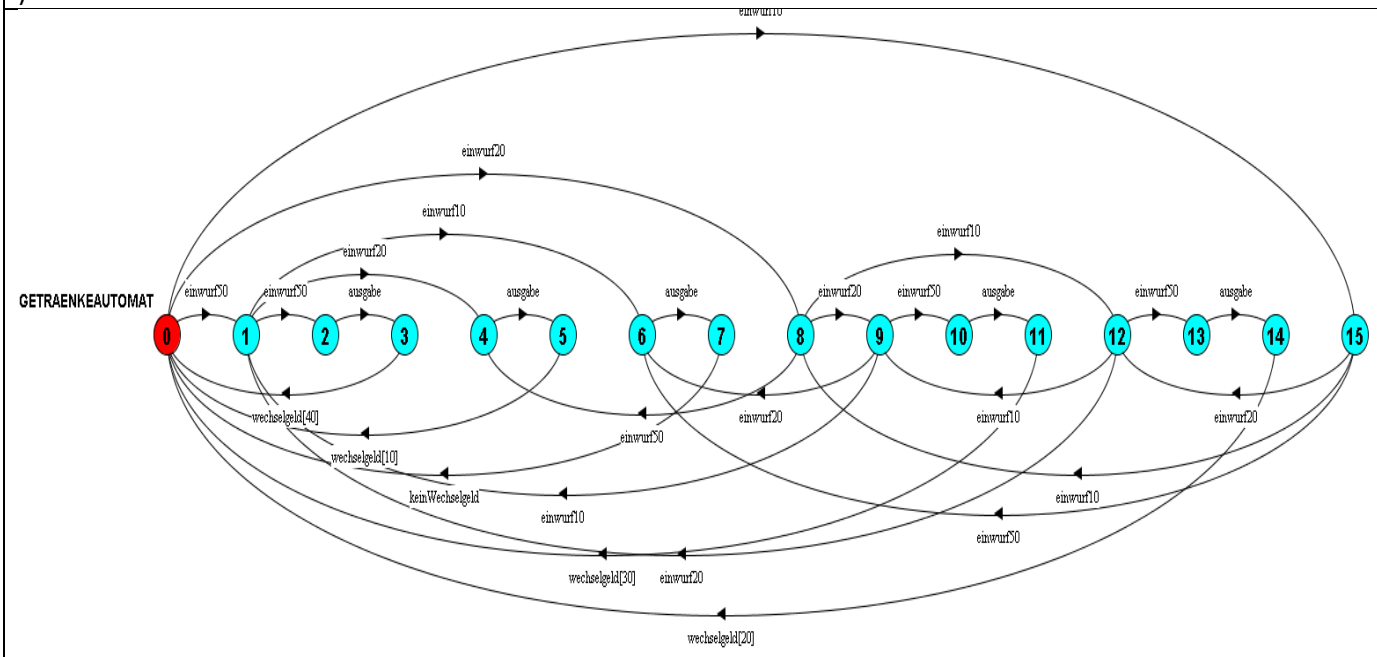
```
range P = 0..9
SENSOR = SENSOR[5],
SENSOR[i:P] = (
   when (i >= 0 && i <= 2) niedrig -> SENSOR[i]
   |when (i >= 8 && i <= 9) hoch -> SENSOR[i]
   |when (i >= 3 && i <= 7) normal -> SENSOR[i]
   |pegel[j:P] -> SENSOR[j]
).
```



4.3

In einem Getränkeautomaten kostet ein Becher süße Brause 60 Cent. Der Automat nimmt 10-, 20- und 50-Cent-Münzen an und gibt gegebenenfalls Wechselgeld zurück. Beschreiben Sie den Getränkeautomaten algebraisch durch FSP und geben Sie den entsprechenden Zustandsgraphen an.

```
const PRICE = 60
const MAX_SUM = PRICE + 50
range N = 0..MAX_SUM
GETRAENKEAUTOMAT = AUTOMAT[0],
AUTOMAT[s:N] = (
   when (s < PRICE && s + 10 <= MAX_SUM) einwurf10 -> AUTOMAT[s + 10]
   |when (s < PRICE && s + 20 <= MAX_SUM) einwurf20 -> AUTOMAT[s + 20]
   |when (s < PRICE && s + 50 <= MAX_SUM) einwurf50 -> AUTOMAT[s + 50]
   |when (s >= PRICE) ausgabe -> WECHSELGELD[s]
),
WECHSELGELD[s:N] = (
   when (s > PRICE) wechselgeld[s - PRICE] -> AUTOMAT[0]
   |when (s == PRICE) keinWechselgeld -> AUTOMAT[0]
).
```



4.4

a) Zeigen Sie, dass die im Folgenden definierten Prozesse `S1` und `S2` das gleiche Verhalten zeigen:

```
P = (a -> b -> P).
Q = (c -> b -> Q).
||S1 = (P || Q).

S2 = ( a -> c -> b -> S2
     | c -> a -> b -> S2).
```

Bei S1 haben P und Q beide den Prozess b, und laut FSP müssen beide Prozesse b synchronisieren.
Mögliche Aktionsfolgen:
P führt a aus, Q führt c aus, dann synchronisieren sie b.

- a -> c -> b

Q führt c aus, P führt a aus, dann synchronisieren sie b

- c -> a -> b

Danach kehren beide zum Anfangszustand zurück. Somit sind mögliche Traces von S1:

- a -> c -> b -> …
- c -> a -> b -> …

Bei S2 gibt es zwei Alternativen
Alternativ 1:
- a -> c -> b -> …

Alternativ 2:
- c -> a -> b -> …

Mögliche Traces von S2:
- a -> c -> b -> …
- c -> a -> b -> …

Das nochmal in Vergleich mit möglichen Traces von S1:
- a -> c -> b -> …
- c -> a -> b -> …

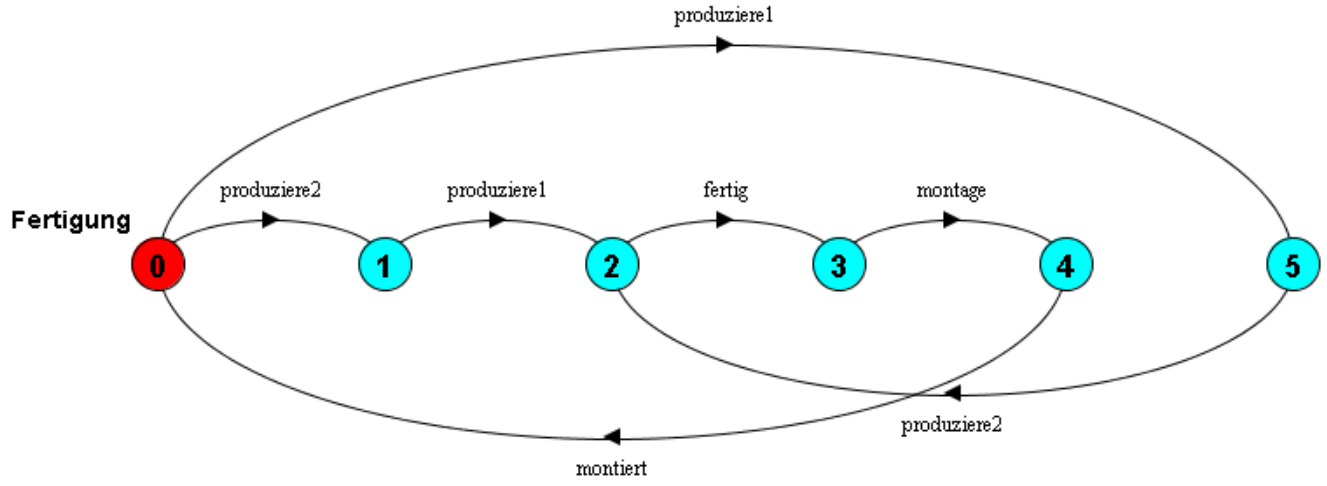
Beide Prozesse haben identische Traces, somit zeigen beide dasselbe Verhalten.

b) Finden Sie einen Prozess, der ohne die Verwendung der parallelen Komposition || definiert ist, aber das gleiche Verhalten wie der folgende Prozess `Fertigung` zeigt.

```
Prod1 = (prod1 -> fertig -> montiert -> Prod1).
Prod2 = (prod2 -> fertig -> montiert -> Prod2).
||Produktion = (Prod1 || Prod2).

Montage = (fertig -> montage -> montiert -> Montage).
||Fertigung = (Produktion || Montage).
```

```
range Produktion = 0..1
range Montage = 0..1
Fertigung = Fertigung[0][0][0],
Fertigung[prod1:Produktion][prod2:Produktion][montageStatus:Montage] = (
        when(prod1 == 0) produziere1 -> Fertigung[1][prod2][montageStatus]
        |when(prod2 == 0) produziere2 -> Fertigung[prod1][1][montageStatus]
        |when(montageStatus == 0 && prod1 == 1 && prod2 == 1) fertig -> montage -> Fertigung[prod1][prod2][1]
        |when(montageStatus == 1 && prod1 == 1 && prod2 == 1) montiert -> Fertigung[0][0][0]
).
```



4.5

Der Prozess `Speicher = (rein -> raus -> Speicher)`. modelliert das Verhalten einer Speicherzelle, indem er zunächst eine `rein`- und dann eine `raus`-Aktionen ausführt.

Definieren Sie unter Verwendung von `Speicher` und mit Hilfe der parallelen Komposition einen Prozess `Schieber`, der das Verhalten eines Schieberegisters mit $N = 4$ internen Speicherzellen modelliert – d. h. das Schieberegister besitzt nach außen nur die Aktionen `rein` bzw. `raus`; im Innern des Schieberegisters werden die Daten von einer Zelle zur nächsten weitergeschoben.

Zeichnen Sie zuerst ein Strukturdiagramm und testen Sie dann Ihre Variante für $N \leq 4$ im Analysetool LTSA.

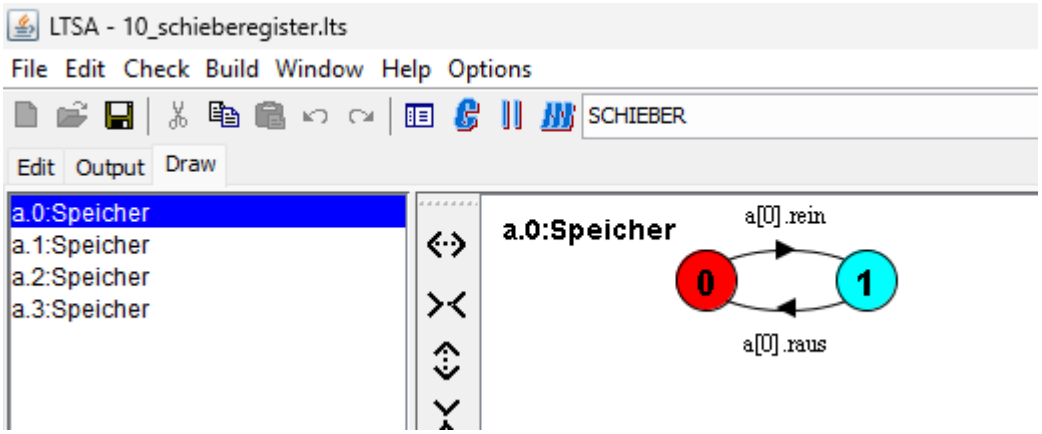
Hinweis: In FSP kann der Befehl `forall` sowohl in der parallelen Komposition als auch beim Relabelling verwendet werde.

---

const N = 4 // Anzahl der internen Speicherzellen
range ID = 0..N-1 // ID
Speicher = (rein -> raus -> Speicher). // Prozess Speicher
||SCHIEBER = forall [i:ID] a[i]:Speicher. // Parallele Komposition der Speicherzellen. Es werden vom Prozess Speicher vier Instanzen erzeugt, jeweils mit einem Label-Präfix a[i] versetzt.
SCHIEBEREGISTER = SCHIEBER
/{ rein/a[0].rein, // Äußere Eingabe mit der ersten Zelle verbinden
raus/a[N-1].raus, // Äußere Ausgabe mit der letzten Zelle verbinden
a[0].raus/a[1].rein, // Verbindung von Zelle 0 zu Zelle 1
a[1].raus/a[2].rein, // Verbindung von Zelle 1 zu Zelle 2
a[2].raus/a[3].rein // Verbindung von Zelle 2 zu Zelle 3
}.

```
const N = 4 // Anzahl der internen Speicherzellen
range ID = 0..N-1   // ID
Speicher = (rein -> raus -> Speicher).  // Prozess Speicher
||SCHIEBER = forall [i:ID] a[i]:Speicher.   // Parallele Komposition der Speich
SCHIEBEREGISTER = SCHIEBER
    /{  rein/a[0].rein, // Äußere Eingabe mit der ersten Zelle verbinden
        raus/a[N-1].raus,   // Äußere Ausgabe mit der letzten Zelle verbinden
        a[0].raus/a[1].rein,    // Verbindung von Zelle 0 zu Zelle 1
        a[1].raus/a[2].rein,    // Verbindung von Zelle 1 zu Zelle 2
        a[2].raus/a[3].rein     // Verbindung von Zelle 2 zu Zelle 3
     }.
```

// Jede Speicherzelle im Schieberegister ist eine Instanz des gegebenenen Prozesses Speicher, der das Verhalten einer einzelnen Speicherzelle modelliert.
// Durch die parallele Komposition von vier Speicherzellen haben wir ein Schieberegister mit vier internen Speicherzellen erstellt.
// Das Schieberegister besitzt nach außen nur die Aktionen rein als Eingabe und raus als Ausgabe wie in der Aufgabenstellung gefordert
// Die Daten werden intern von einer Zelle zur nächsten weiterverschoben, indem die raus-Aktion einer Zelle mit der rein-Aktion der nächsten Zelle verbunden wird.
// Das Modell kann durch ein Strukturdiagramm dargestellt werden, in dem die vier Speicherzellen in Reihe geschaltet sind
// und die Daten von der ersten zur letzten Zelle wandern.

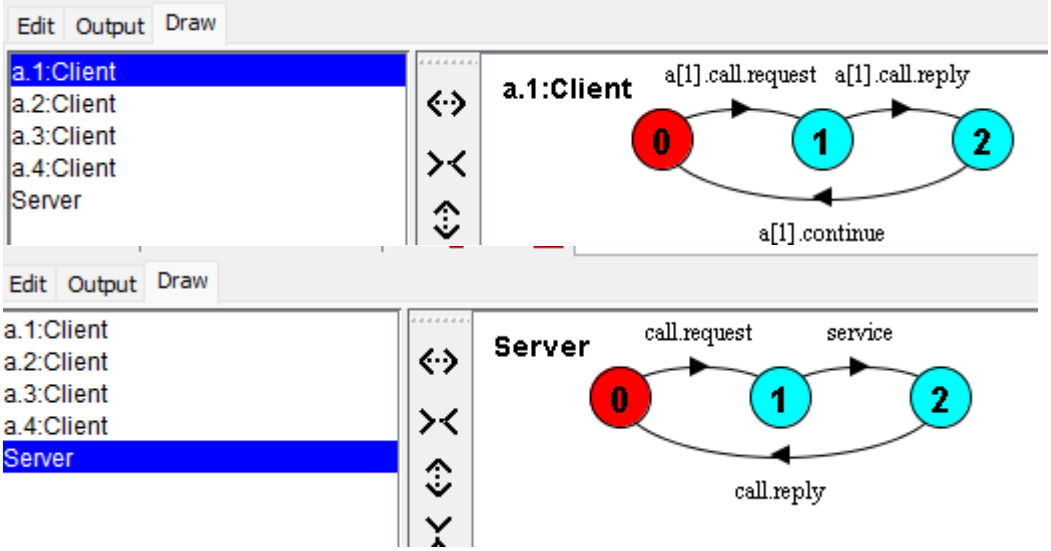


Ich habe bei dieser Aufgabenstellung das Problem, dass ich keinen Prozess Schieber bekomme, sondern mehrere Prozesse Speicher. Ich habe vieles versucht, aber ich bekomme einfach nicht weiter. Gibt es hierzu eventuell eine Musterlösung? Können Sie vielleicht meinen Fehler erkennen?

4.6

a) Modifizieren Sie den Client-/Server-Prozess aus der Vorlesung so, dass mehr als ein Client den Dienst des Servers abrufen kann.

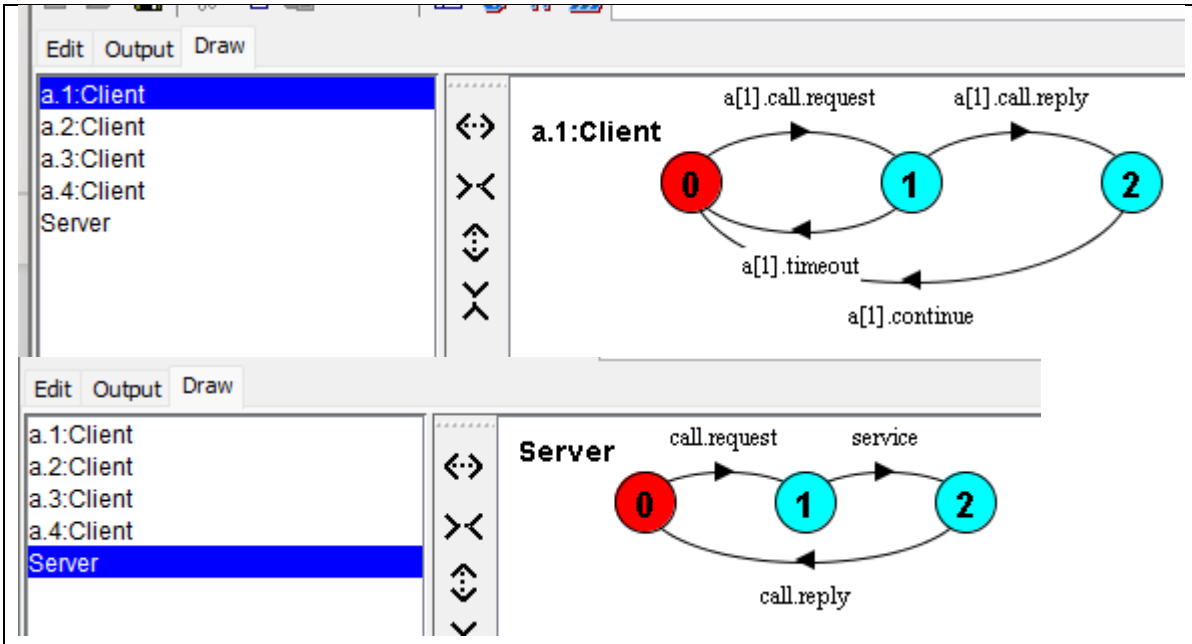
```
Client = (call.request -> call.reply -> continue -> Client).
Server = (accept.request -> service -> accept.reply -> Server).
||ClientServer(N=4) = (a[i:1..N]:Client || Server)
/ {call.request/accept.request, call.reply/accept.reply}.
```



Bei dieser Aufgabenstellung habe ich ein ähnliches Problem. Ich habe soweit ich weiß, zusammenhängenden Aktionen synchronisiert, aber einen Prozess ClientServer bekomme ich nicht, sondern mehrere Einzelprozesse

b) Ändern Sie Ihren Prozess aus dem ersten Aufgabenteil so, dass die `call`-Aktion der Clients statt mit einer Antwort des Servers auch mit einem Timeout beendet werden kann.

```
Client = (call.request -> (call.reply -> continue -> Client | timeout -> Client)).
Server = (accept.request -> service -> accept.reply -> Server).
||ClientServer(N=4) = (a[i:1..N]:Client || Server)
/ {call.request/accept.request, call.reply/accept.reply}.
```



Hier habe ich dasselbe Problem

4.7

Ein Museum besitzt einen Eingang und einen Ausgang, die jeweils durch ein Drehkreuz gesichert und mit einer zentralen Steuerung verbunden sind. Zur Öffnungszeit gibt der Museumswärter an der zentralen Steuerung den Eingang ins Museum frei, woraufhin die beiden Drehkreuze entsperrt werden. Nach Ablauf der Besuchszeit sperrt er an der zentralen Steuerung den Eintritt weiterer Besucher. Nach der Schließung kann nur noch das Drehkreuz am Ausgang passiert werden (solange sich noch Besucher in den Räumen befinden). Da es sich um ein sehr kleines Museum handelt, erlaubt die Steuerung auch nur N Besuchern gleichzeitig, sich in den Räumen aufzuhalten.

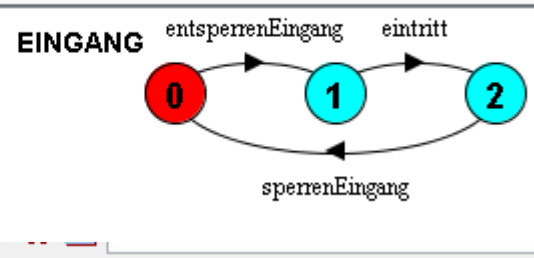
Modellieren Sie das Museum durch die parallele Komposition der Prozesse `Eingang`, `Ausgang`, `Steuerung` und `Waerter`.

Edit Output Draw

```
const N = 5
range B = 0..N
EINGANG = (
    entsperrenEingang -> eintritt -> sperrenEingang -> EINGANG
).
AUSGANG = (
    entsperrenAusgang -> austritt -> sperrenAusgang -> AUSGANG
).
STEUERUNG = STEUERUNG[0],
STEUERUNG[b:B] = (
    when(b < N) eintritt -> STEUERUNG[b + 1]
    |when(b > 0) austritt -> STEUERUNG[b - 1]
    |oeffnenSteuerung -> STEUERUNG[b]
    |schliessenSteuerung -> STEUERUNG[b]
).
WAERTER = (
    oeffneEingang -> oeffnenSteuerung -> warteBesuchszeit -> schliesseEingang -> schliessenSteuerung -> WAERTER
).
||MUSEUM = (EINGANG || AUSGANG || STEUERUNG || WAERTER)
/{
    entsperrenEingang/oeffnenSteuerung,
    sperrenEingang/schliessenSteuerung,
    entsperrenAusgang/oeffnenSteuerung,
    sperrenAusgang/schliessenSteuerung,
    eintritt/eintritt,
    austritt/austritt
}.
```

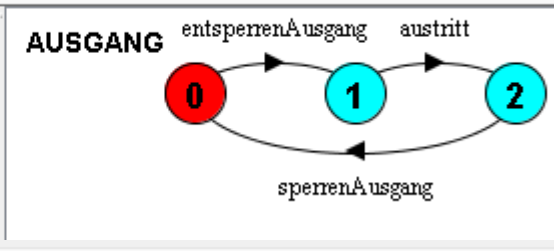
Edit Output Draw

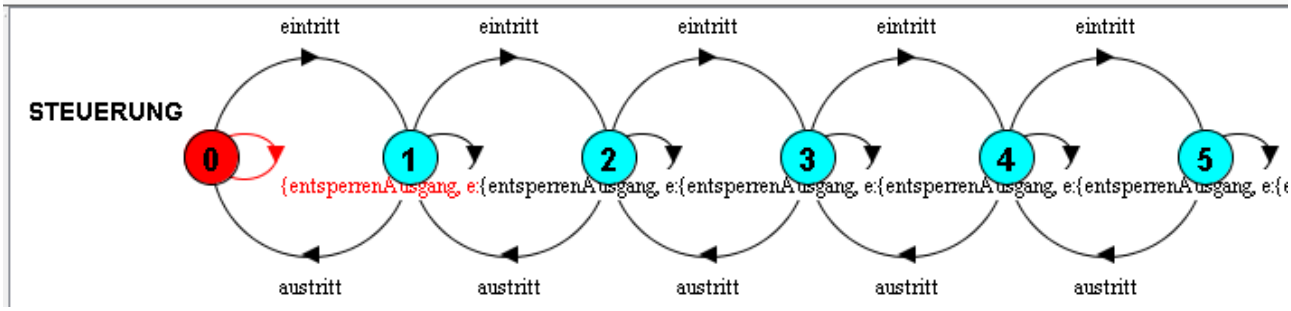
EINGANG
AUSGANG
STEUERUNG
WAERTER



Edit Output Draw

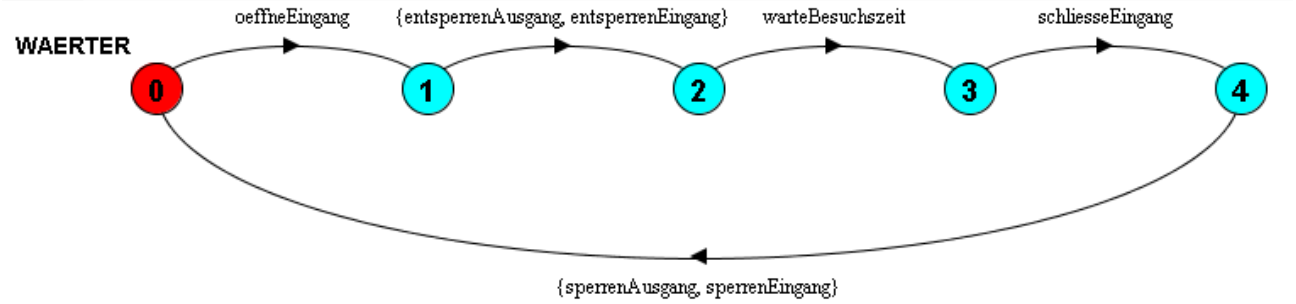
EINGANG
AUSGANG
STEUERUNG
WAERTER





In geschweiften Klammern steht: {entsperrenAusgang, entsperrenEingang, sperrenAusgang, sperrenEingang}

Compile



Ich habe bei dieser Aufgabenstellung auch das Problem, dass ich keinen Prozess MUSEUM bekomme, sondern nur die Einzelprozesse. Ich habe versucht durch Relabelling dieselben Aktionen zu synchronisieren.
Hier nochmal der Code per Text:

```
const N = 5
range B = 0..N
EINGANG = (
   entsperrenEingang -> eintritt -> sperrenEingang -> EINGANG
).
AUSGANG = (
   entsperrenAusgang -> austritt -> sperrenAusgang -> AUSGANG
).
STEUERUNG = STEUERUNG[0],
STEUERUNG[b:B] = (
   when(b < N) eintritt -> STEUERUNG[b + 1]
   |when(b > 0) austritt -> STEUERUNG[b - 1]
   |oeffnenSteuerung -> STEUERUNG[b]
   |schliessenSteuerung -> STEUERUNG[b]
).
WAERTER = (
   oeffneEingang -> oeffnenSteuerung -> warteBesuchszeit -> schliesseEingang -> schliessenSteuerung -> WAERTER
).
||MUSEUM = (EINGANG || AUSGANG || STEUERUNG || WAERTER)
/{
   entsperrenEingang/oeffnenSteuerung,
   sperrenEingang/schliessenSteuerung,
   entsperrenAusgang/oeffnenSteuerung,
   sperrenAusgang/schliessenSteuerung,
   eintritt/eintritt,
   austritt/austritt
}.
```