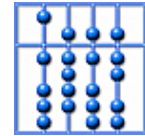


---

## **Vorlesung Specification of Distributed Systems**

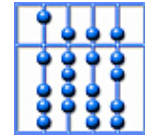
Dr. Bernhard Schätz  
Leopold-Franzens Universität Innsbruck  
Sommersemester 2005



## Overview

---

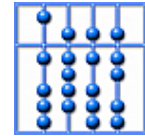
1. Introduction
2. Basics: Behavior, Interaction, Concurrency
3. Coroutines
4. Data Flow Models
5. Communicating Processes
6. State-Based Models
7. Coordination
8. Executions
9. Property Descriptions



## Overview

---

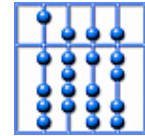
1. Introduction
2. Basics: Behavior, Interaction, Concurrency
3. Coroutines
4. Data Flow Models
5. Communicating Processes
  1. Sequential Processes
  2. Concurrent Processes
6. State-Based Models
7. Coordination
8. Executions
9. Property Descriptions



## Questions

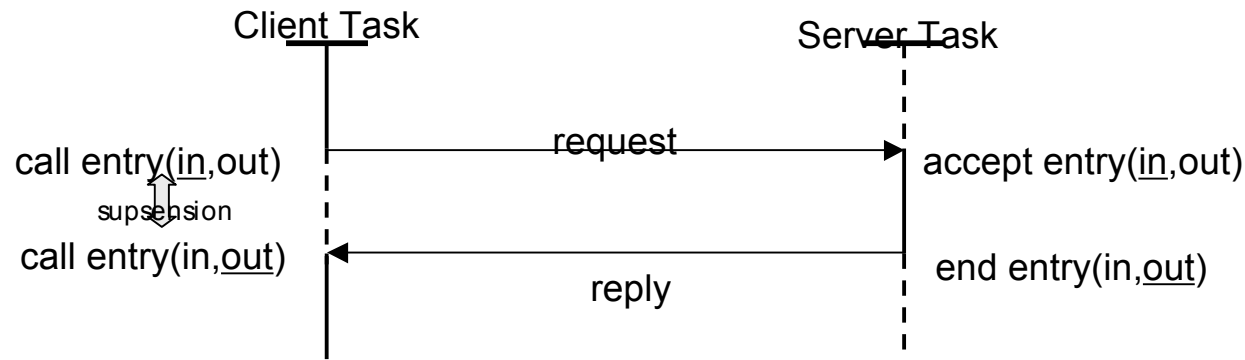
---

1. Exercise: Model two processes communicating via the Ada rendezvous concept using CSP.
2. Exercise: Model a Java implementation of the parking lot example with the counter as a synchronized object, using a CSP formalization.
3. How can a process-algebraic description turned over into a labeled transition system? What are the states? What are the transitions?
4. What are the interaction traces of the system  $P1 \parallel P2 \parallel \text{Merge} \parallel C$  with  $P1 = p1(x) \rightarrow s1(x) \rightarrow P1$ ,  $P2 = p2(x) \rightarrow s2(x) \rightarrow P2$ ,  $\text{Merge} = s1(x) \rightarrow r(x) \rightarrow \text{Merge} \mid s2(x) \rightarrow r(x) \rightarrow \text{Merge}$ , and  $C = r(x) \rightarrow c(x) \rightarrow C$  ?
5. What is the corresponding CSP-term for the set of (complete) interaction traces consisting of  $(\text{coin} \cdot \{ (\text{tea-button} \cdot \text{tea}), (\text{coffee-button} \cdot \text{coffee}) \})^\infty$  ?
6. Is there only one corresponding CSP-term (up to use of sub-terms)?
7. With Question 6 in mind, are (complete) interaction traces a good model for CSP processes?
8. Can you think of an extension of interaction traces, supplying a good model for CSP processes?
9. What is the behavior of the process  $B \setminus \{\text{tick}\}$  with  $B = \text{tick} \rightarrow B \mid \text{bang} \rightarrow \text{Stop}$  ?
10. How can  $S = T \parallel U$  for  $S = \mu P. (a \rightarrow P \mid b \rightarrow P)$ ,  $T = (\mu Q. a \rightarrow Q)$ ,  $U = (\mu R. b \rightarrow R)$  be established?



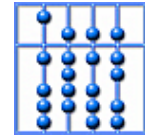
## Exercise 1

1. Model two processes communicating via the Ada rendezvous concept using CSP.



CSP-Model of two processes:

- Client: Process is suspended until rendezvous is performed:  
 $Cl = \dots \rightarrow \text{call\_entry}(\text{in}) \rightarrow \text{end\_entry}(\text{out}) \rightarrow \dots$
- Server: Rendezvous is blocked until server is ready to perform it:  
 $Sv = \dots \rightarrow \text{call\_entry}(\text{in}) \rightarrow \text{end\_entry}(\text{out}) \rightarrow \dots$



## Exercise 2

Model a Java implementation of the parking lot example with the counter as a synchronized object, using a CSP formalization.

```
class InGate extends Thread
{
    ...
    Counter capacity;

    InGate(Counter c)
        { capacity = c; }

    public void run() {

        ...
        while (true){
            ...
            capacity.decrement();
            ...
        }
    }
}
```

```
class OutGate extends Thread
{
    ...
    Counter capacity;

    OutGate(Counter c)
        { capacity = c; }

    public void run() {

        ...
        while (true){
            ...
            capacity.increment();
            ...
        }
    }
}
```

```
class Counter {
    int amount = 0;

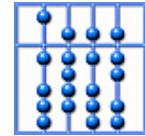
    Counter(int a)
        { amount = a; }

    synchronized void
        increment() {
        this.amount =
            this.amount + 1;
        }

    synchronized void
        decrement() {
        if(this.amount > 0)
            this.amount =
                this.amount - 1;
        }
}
```

CSP-Formalization:

- InGate, OutGate: Model as processes to capture active execution as threads
- Counter: Model as process to capture of synchronized object



## Exercise 2

```
class InGate extends Thread
{
    ...
    Counter capacity;

    InGate(Counter c)
    { capacity = c; }

    public void run() {

        ...
        while (true){
            ...
            capacity.decrement();
            ...
        }
    }
}
```

```
class OutGate extends Thread
{
    ...
    Counter capacity;

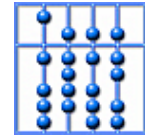
    OutGate(Counter c)
    { capacity = c; }

    public void run() {

        ...
        while (true){
            ...
            capacity.increment();
            ...
        }
    }
}
```

Thread-Formalization:

- InGate: InGate = ... → decrement → ... → InGate
- OutGate: OutGate = ... → increment → ... → OutGate



## Exercise 2

```
class Counter {
  int amount = 0;

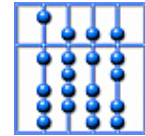
  Counter(int a)
  { amount = a; }

  synchronized void
    increment() {
    this.amount =
      this.amount + 1;
  }

  synchronized void
    decrement() {
    if(this.amount > 0)
      this.amount =
        this.amount - 1;
  }
}
```

Formalization synchronized object:

- Counter(x) =  
(x > 0  $\triangleright$  decrement  $\rightarrow$  Counter(x-1)) |  
(increment  $\rightarrow$  Counter(x+1))



## Question 3

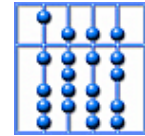
---

How can a process-algebraic description turned over into a labeled transition system? What are the states? What are the transitions?



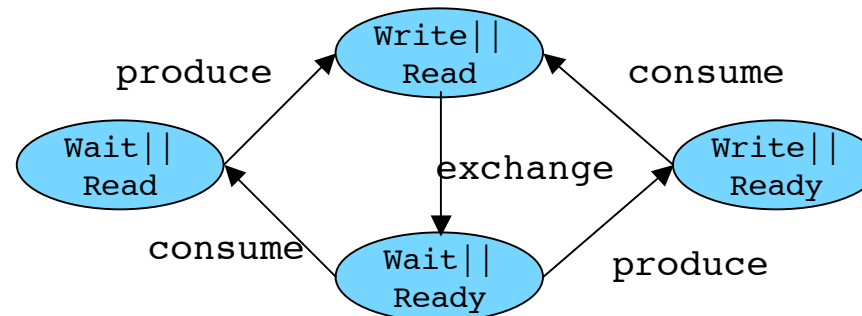
Example: Producer/Consumer:

- Algebra:  $P = \text{Wait}$ ,  $\text{Wait} = \text{produce} \rightarrow \text{Write}$ ,  $\text{Write} = \text{exchange} \rightarrow \text{Wait}$ ,  $C = \text{Read}$ ,  $\text{Read} = \text{exchange} \rightarrow \text{Ready}$ ,  $\text{Ready} = \text{consume} \rightarrow \text{Read}$
- Transition system: States are process terms, transitions are unfoldings of process terms according to laws



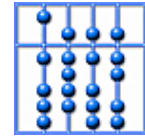
### Question 3

How can a process-algebraic description turned over into a labeled transition system? What are the states? What are the transitions?



Transition system: States are process terms, transitions are unfoldings of process terms according to laws

- States: Wait || Read, Write || Read, Wait || Ready, Write || Ready
- Transitions: e.g.,
  - (Wait || Read, produce, Write || Read) via “Wait || Read = produce → Write || Read”
  - (Write || Read, exchange, Wait || Ready) via “Write || Read = exchange → Wait || Ready”.



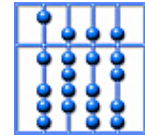
## Question 4

---

What are the interaction traces of the system  $P1 \parallel P2 \parallel \text{Merge} \parallel C$  with  $P1 = p1(x) \rightarrow s1(x) \rightarrow P1$ ,  $P2 = p2(x) \rightarrow s2(x) \rightarrow P2$ ,  $\text{Merge} = s1(x) \rightarrow r(x) \rightarrow \text{Merge} \mid s2(x) \rightarrow r(x) \rightarrow \text{Merge}$ , and  $C = r(x) \rightarrow c(x) \rightarrow C$  ?

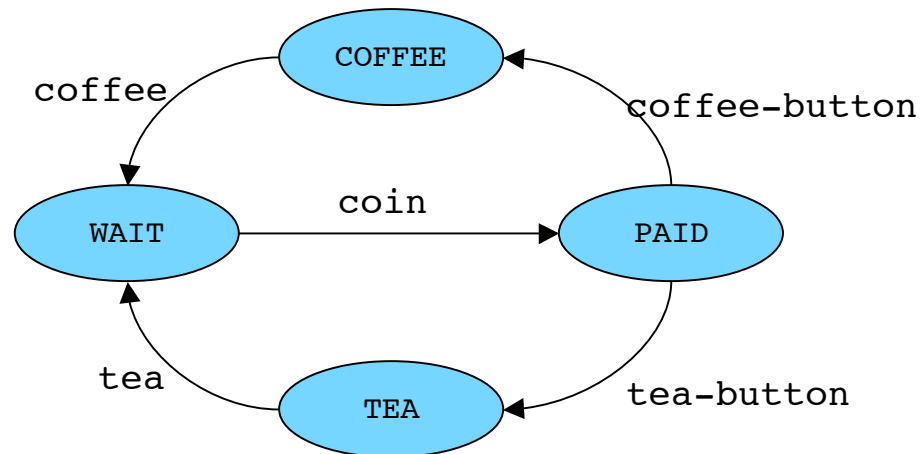
Observation traces: (assuming integer  $x$ )

- Formally:  $\text{Traces}(P) = \{a \cdot t \mid a \rightarrow P' \text{ and } t \in \text{Traces}(P')\}$
- Examples:
  - $\langle \rangle$
  - $p1(1) \cdot s1(2) \cdot p2(2) \cdot r(1)$
  - $p1(1) \cdot s1(2) \cdot p2(2) \cdot r(1) \cdot p1(3) \cdot c(1) \cdot s2(2) \cdot r(2) \cdot c(2)$
  - $p1(1) \cdot s1(2) \cdot p2(2) \cdot r(1) \cdot p1(3) \cdot c(1) \cdot s2(2) \cdot r(2) \cdot c(2) \cdot s1(3) \cdot r(3) \cdot c(3)$



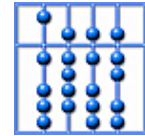
## Question 5

What is the corresponding CSP-term for the set of (complete) interaction traces consisting of  $(\text{coin} \cdot \{ (\text{tea-button} \cdot \text{tea}), (\text{coffee-button} \cdot \text{coffee}) \})^\infty$  ?



Corresponding process term:

- Machine =  $\text{coin} \rightarrow \text{Paid}$
- Paid =  $(\text{tea-button} \rightarrow \text{Tea}) \mid (\text{coffee-button} \rightarrow \text{Coffee})$
- Tea =  $\text{tea} \rightarrow \text{Machine}$
- Coffee =  $\text{coffee} \rightarrow \text{Machine}$



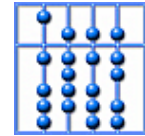
## Question 6

---

Is there only one corresponding CSP-term (up to use of sub-terms)? Justify your answer!

No! There is an automaton with the same set of traces:

- $\text{StubbornMachine} = \text{coin} \rightarrow \text{SellTea} \mid \text{coin} \rightarrow \text{SellCoffee}$
- $\text{SellTea} = \text{tea-button} \rightarrow \text{Tea}$
- $\text{SellCoffee} = \text{coffee-button} \rightarrow \text{Coffee}$
- $\text{Tea} = \text{tea} \rightarrow \text{StubbornMachine}$
- $\text{Coffee} = \text{coffee} \rightarrow \text{StubbornMachine}$



## Question 7

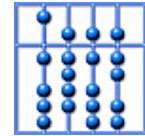
---

With Question 6 in mind, are (complete) interaction traces a good model for CSP processes? Justify your answer!

Incomplete interaction traces are unsuitable because they cannot distinguish between blocking and non-blocking processes. Complete interaction traces are not a suitable model because it is not compositional!

Explanation:

- Consider the following customer:  
Customer = coin  $\rightarrow$  tea-button  $\rightarrow$  tea  $\rightarrow$  Customer
- Customer || Machine will never block; it is equivalent to Customer
- Customer || StubbornMachine may block after every coin-action; it is equivalent to  
SellerStop = (coin  $\rightarrow$  tea-button  $\rightarrow$  tea  $\rightarrow$  SellOrStop) | (coin  $\rightarrow$  STOP)
- Therefore: Although Machine and StubbornMachine have the same traces, when combined with Customer they lead to different sets of traces.



## Question 8

---

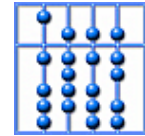
Can you think of an extension of interaction traces, supplying a good model for CSP processes?

Refusal traces  $(t,R)$ :

- Trace  $t$ : Observation trace a process may engage in
- Set of actions  $R$ : Actions a process may refuse after engaging in  $t$

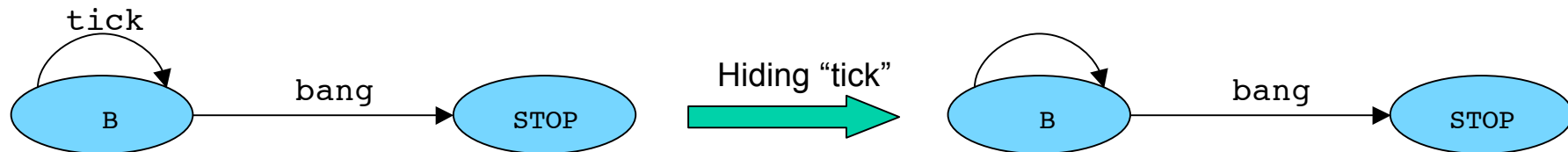
Refusal traces of machines:

- Machine:  $\{ (<>, \{\text{tea-button, coffee-button, coffee, tea}\}), (<>, \{\text{tea-button, coffee}\}), (\text{coin}, \{\text{coin, coffee, tea}\}), (\text{coin}, \{\text{tea}\}), (\text{coin} \bullet \text{tea-button}, \{\text{coin, tea-button, coffee-button, coffee}\}), \dots \}$
- StubbornMachine:  $\{ (<>, \{\text{tea-button, coffee-button, coffee, tea}\}), (<>, \{\text{tea-button, coffee}\}), (\text{coin}, \{\text{coin, coffee, tea, **coffee-button**\}), (\text{coin}, \{\text{coin, coffee, tea, **tea-button**\}), (\text{coin}, \{\text{tea}\}), (\text{coin} \bullet \text{tea-button}, \{\text{coin, tea-button, coffee-button, coffee}\}), \dots \}$



## Question 9

What is the behavior of the process  $B \setminus \{\text{tick}\}$  with  $B = \text{tick} \rightarrow B \mid \text{bang} \rightarrow \text{Stop}$  ?

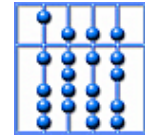


Hiding “tick”: Two possible unfoldings:

- $B \setminus \{\text{tick}\} \Rightarrow (\text{tick} \rightarrow B \mid \text{bang} \rightarrow \text{STOP}) \setminus \{\text{tick}\} \Rightarrow (\text{tick} \rightarrow B) \setminus \{\text{tick}\} \Rightarrow B \setminus \{\text{tick}\}$
- $B \setminus \{\text{tick}\} \Rightarrow (\text{tick} \rightarrow B \mid \text{bang} \rightarrow \text{STOP}) \setminus \{\text{tick}\} \Rightarrow (\text{bang} \rightarrow \text{STOP}) \setminus \{\text{tick}\} \Rightarrow \text{bang} \rightarrow (\text{STOP} \setminus \{\text{tick}\}) \Rightarrow \text{bang} \rightarrow \text{STOP}$

Interpretation:

- $B \setminus \{\text{tick}\}$  may loop forever unnoticed or eventually issue action bang.
- Formally:  $B \setminus \{\text{tick}\}$  may *diverge*



## Question 10

---

How can  $S = V$  with  $V = T \parallel U$  for  $S = \mu P. (a \rightarrow P \mid b \rightarrow P)$ ,  $T = (\mu Q. a \rightarrow Q)$ ,  $U = (\mu R. b \rightarrow R)$  be established?

To establish this equality, a “higher-order” inductive law is needed:

- $P = F(P) \Rightarrow P = \mu Q. F(Q)$  (if  $F$  ensures prefixed actions during recursion)

Then:  $V$

$$= T \parallel U$$

$$= (\mu Q. a \rightarrow Q) \parallel (\mu R. b \rightarrow R)$$

$$= (a \rightarrow (\mu Q. a \rightarrow Q)) \parallel (b \rightarrow (\mu R. b \rightarrow R))$$

$$= ((a \rightarrow T) \parallel (b \rightarrow U))$$

$$= (a \rightarrow (T \parallel (b \rightarrow U))) \parallel (b \rightarrow (a \rightarrow T) \parallel U)$$

$$= (a \rightarrow (T \parallel U)) \mid (b \rightarrow (T \parallel U))$$

$$= (a \rightarrow V) \mid (b \rightarrow V)$$

Inductive reasoning:  $V = (a \rightarrow V) \mid (b \rightarrow V) \Rightarrow V = \mu P. (a \rightarrow P) \mid (b \rightarrow P)$

Thus:  $V = S$