

Course “Modelling of Concurrent Systems”  
Summer Semester 2019  
University of Duisburg-Essen

Harsh Beohar  
LF 265,  
harsh.beohar@uni-due.de

# Course handler

## Harsh Beohar

- Room LF 265
- E-Mail: `harsh.beohar@uni-due.de`
- Meeting by appointment.
- Please send mail only by your official student mail id's.  
`http://www.uni-due.de/zim/services/e-mail/`

Task: Lecturer + Exercise Tutor.

**Website:** `https://www.uni-due.de/theoinf/teaching/ss2019\_modns.php`

# Lecture schedule

## Schedule:

- Tuesday, 14:15-15:30, in Room LF 035
- Friday, 12:15-13:30, in Room LF 035

# Exercises

## Schedule:

(Roughly, every fourth lecture kept for exercises.)

- Tuesday, 30/04, 14:15-15:30, in Room LF 035.
- Tuesday, 14/05, 14:15-15:30, in Room LF 035.
- Tuesday, 28/05, 14:15-15:30, in Room LF 035.
- Friday, 14/06, 12:15-13:30, in Room LF 035.
- Yet to be decided.

## Idea:

- Problem sheet will be announced in the class, whenever it is published.
- At the same time, also the deadline to submit the exercises will also be announced.
- Submission: in person or email. Please use Matriculation number and sheet number.

# Exercises

## Scheme:

- If the sum is more than 60% then you get a bonus point.
- Effect is improvement by one grade level. E.g. 2.3 to 2.0
- Group solutions are not allowed.

# Target audience

## MAI

Master “Applied computer science” (“Angewandte Informatik”) - focus engineering or media computer science:

- In the brochure you can find the field of application:  
“Distributed Reliable Systems” (“Verteilte, Verlässliche Systeme”)  
Concurrent systems (Nebenläufige Systeme)

Stundenzahl: 4 SWS (3V + 1Ü), 6 Credits

# Target audience

## Master ISE/CE – Verteilte, Verlässliche Systeme

In Master “ISE – Computer Engineering”, this lecture is classified as follows:

- Elective “Verteilte, Verlässliche Systeme”  
(Reliable Systems)  
Stundenzahl: 4 SWS (3V + 1Ü)

# Requirement

Prerequisites:

- Automata and Formal languages.
- Logic.

For the past teaching content, see

<http://www.ti.inf.uni-due.de/teaching/ss2018/mod-ns/>



# Examination

The exam will be held as a viva voce (oral examination).

Planned dates:

18th July 2019 (Thursday) and 19th July 2019 (Friday).

# Literature

- Jos Baeten, Twan Basten, and Michel Reniers.  
*Process algebra: Equational theories of communicating processes* Cambridge University Press, 2010.  
 Contents: **(Probabilistic) Process algebra.**
- Luca Aceto, Anna Ingólfssdóttir, Kim G. Larsen, Jiri Srba.  
*Reactive Systems: Modelling, Specification and Verification.*  
 Cambridge University Press, 2007.  
 Contents: **Strong and weak bisimulation, Hennessy-Milner logic, Timed automata**
- Rob van Glabbeek.  
*Behavioural equivalences.* The linear time - branching time spectrum I. In proc. of CONCUR 90.  
 Contents: **Failure equivalence, (bi)simulation equivalence**

# Literature in Process algebra

- Robin Milner.

*Communication and Concurrency*. Prentice Hall, 1989.

Contents: **Process calculus (CCS), Strong and weak bisimulation, Hennessy-Milner logic.**

- Tony Hoare.

*Communicating sequential processes* 2004. Available at <http://www.usingcsp.com/cspbook.pdf>

Contents: **Process calculus CSP, Failure equivalence**

- Bill Roscoe.

*The Theory and Practice of Concurrency*

1997. Available at

<http://www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf>.

Contents: **A reference book on CSP**

# Literature in Hybrid systems

- Hybrid process algebra by Pieter Cuijpers.

Elsevier journal article or PhD thesis under the same name.

Contents: **Hybrid transition systems, hybrid process algebra, stateless bisimulation**

- André Platzer.

*Logical foundations of Cyber-Physical Systems*. Springer, 2018.

Contents: Semantics of differential equations, differential logic.

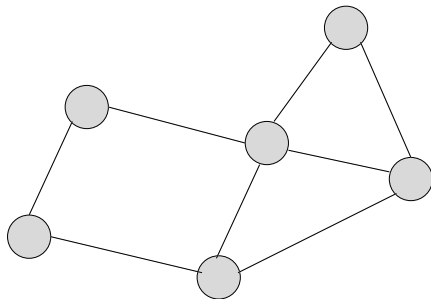
# Lecture style

- We will follow “the” process algebra book. Also, which sections are to be read for the next lecture will be announced in the current one.
- Very few materials will be presented using slides and mostly on blackboard. So please make your own notes!
- However, for hybrid systems there will be lecture notes.

# Motivation

What are concurrent systems?

**In general:** systems in which several components/processes run concurrently and typically communicate via message passing.



# Motivation

Concurrency versus parallelism:

# Motivation

Concurrency versus parallelism:

## Parallelism

Two events take place in **parallel** if they are executed at the same moment in time.

## Concurrency

Two events are **concurrent** if they *could potentially* be executed in parallel, but they do not have to. This means there is no causal dependency between them.



# Motivation

Concurrency versus parallelism:

## Parallelism

Two events take place in **parallel** if they are executed at the same moment in time.

## Concurrency

Two events are **concurrent** if they *could potentially* be executed in parallel, but they do not have to. This means there is no causal dependency between them.

Hence: concurrency is the more general term.

Examples?

# Motivation

## (Potential) characteristics of concurrent systems

- Concurrency/parallelism
- Openness (extendability, interaction with the environment)
- Modularity
- Nonterminating behaviour (infinite runs)
- Nondeterminism
- Temporal properties (e.g. “an event will occur eventually”)

# Motivation

## Problems with concurrent systems

- Deadlocks
- Guaranteeing mutual exclusion
- Infinite respectively huge state space
- Strongly dynamic behaviour/changing number of processes
- Variable topology/mobility

# Motivation

## Problems with concurrent systems

- Deadlocks
- Guaranteeing mutual exclusion
- Infinite respectively huge state space
- Strongly dynamic behaviour/changing number of processes
- Variable topology/mobility

**Hence:** We need methods to model, analyze and verify such systems.

# Change in view

## Classic view

- Program is a function that transform an input into output.
- Two programs are equivalent if and only if they compute the same output for every input.

# Change in view

## Classic view

- Program is a function that transform an input into output.
- Two programs are equivalent if and only if they compute the same output for every input.

- 1 `x = 1;`
- 2 `x = x + 1;`
- 3 `print x;`

# Change in view

## Classic view

- Program is a function that transform an input into output.
- Two programs are equivalent if and only if they compute the same output for every input.

①  $x = 1;$

②  $x = x + 1;$

③  $\text{print } x;$

①  $x = 1;$

②  $x = x \times 2;$

③  $\text{print } x;$

# Change in view

## Classic view

- Program is a function that transform an input into output.
- Two programs are equivalent if and only if they compute the same output for every input.

①  $x = 1;$

②  $x = x + 1;$

③  $\text{print } x;$

||

①  $x = 1;$

②  $x = x \times 2;$

③  $\text{print } x;$

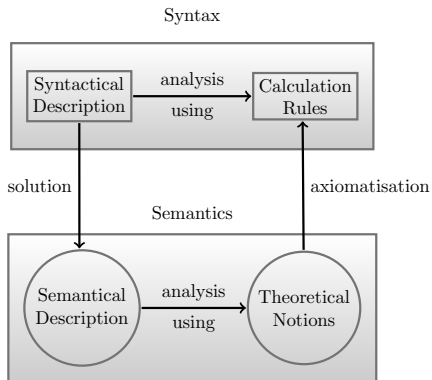


# Mathematical modelling

- Inspired from traditional engineering disciplines.
- Make system models in formal way.
- Analyse them.
- Then build the 'real' system and test it against those models.

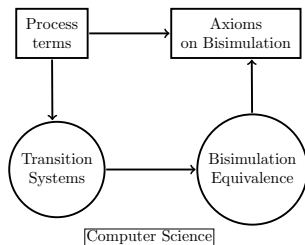
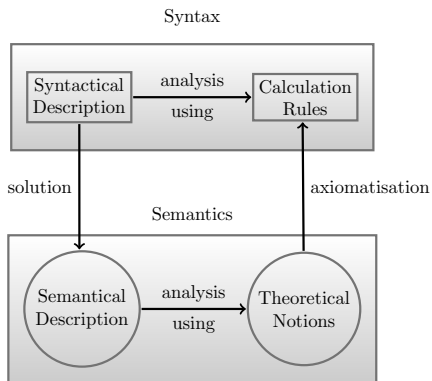
# Mathematical modelling (Cuijpers 2004.)

Syntax aims at concise and finite way of handling semantical notions, which can be infinite mathematical objects.



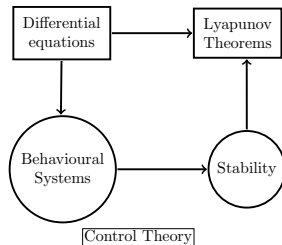
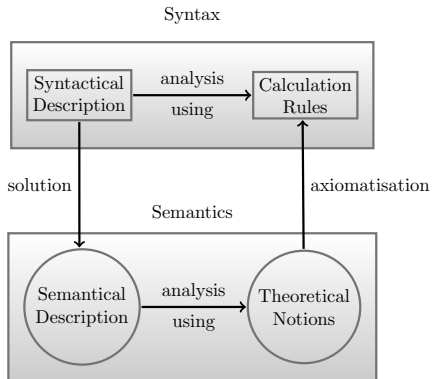
# Mathematical modelling (Cuijpers 2004.)

Syntax aims at concise and finite way of handling semantical notions, which can be infinite mathematical objects.



# Mathematical modelling (Cuijpers 2004.)

Syntax aims at concise and finite way of handling semantical notions, which can be infinite mathematical objects.



# Table of contents

We will introduce the following models for concurrent systems:

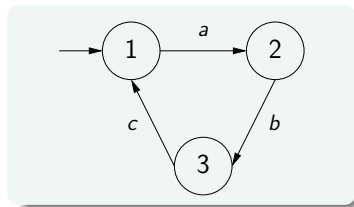
- Transition systems
- Models which are closer to realistic programming languages (for instance process calculi)
- Additional models: Hybrid transition systems.

Furthermore (in order to investigate/analyze systems):

- Specification of properties of concurrent systems (Hennessy-Milner logics)
- Behavioural equivalences: When do two systems behave the same (from the point of view of an external observer)?

# Transition systems

- Transition systems represent states and transitions between states.
- True parallelism is not directly represented.
- Strong similarity to automata, however we are here not so much interested in the accepted language.



# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a

# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a reflexive and transitive relation.



# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a reflexive and transitive relation.
- A **partial order** (poset)  $R$  is a

# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a reflexive and transitive relation.
- A **partial order** (poset)  $R$  is a preorder that is antisymmetric.

# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a reflexive and transitive relation.
- A **partial order** (poset)  $R$  is a preorder that is antisymmetric.
- An **equivalence relation**  $R$  is a

# Reviews of some notions

## Definitions

- For a set  $X$ , we write  $X^*$  for the set of all **finite words** including the empty one  $\varepsilon$ .
- For a set  $X$ , we write  $X^\omega$  the set of all **infinite words**. Also, we write  $X^\infty = X^* \cup X^\omega$ .
- A **binary relation**  $R$  between the sets  $X$  and  $Y$  is a subset of  $X \times Y$ , i.e.,  $R \subseteq X \times Y$ .
- Often, we write  $xRy$  iff  $(x, y) \in R$ .
- A **preorder**  $R \subseteq X \times Y$  is a reflexive and transitive relation.
- A **partial order** (poset)  $R$  is a preorder that is antisymmetric.
- An **equivalence relation**  $R$  is a partial order that is symmetric.

# Transition system space

## Formal definition

A **transition system space** is a triple  $(S, L, \rightarrow)$  of

- ① a set of states  $S$ ;
- ② a set of labels  $L$ ;
- ③ a transition relation  $\rightarrow \subseteq S \times L \times S$ ;

Notations:

- $s \xrightarrow{a} t \iff (s, a, t) \in \rightarrow$
- $s \not\xrightarrow{a} \iff \nexists t \in S \ s \xrightarrow{a} t$

# Basics

Example on board.

## Definition

The reachability relation  $\rightarrow \subseteq S \times L^* \times S$  is inductively defined as follows:

$$\frac{}{s \xrightarrow{\epsilon} s} \quad \frac{s \xrightarrow{w} s' \quad s' \xrightarrow{a} s''}{s \xrightarrow{wa} s''}$$

The transition system induced by state  $s$  consists of all states reachable from  $s$ , and it has the transitions and final states induced by the transition system space.

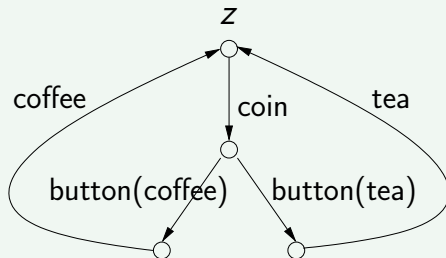
# Transition systems (examples)

A **classical example**: the tea/coffee-machine

We want to model a very simple machine that

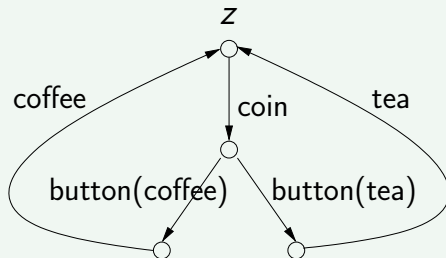
- outputs tea or coffee after a coin has been inserted and a button has been pressed,
- can show faulty behaviour *and*
- may potentially behave non-deterministically.

# Transition systems (examples)



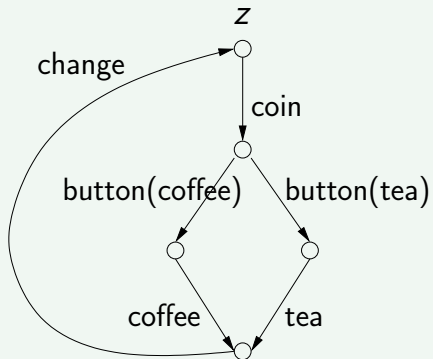


# Transition systems (examples)

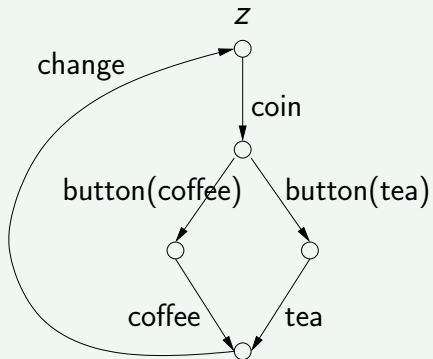


A tea/coffee-machine.

# Transition systems (examples)

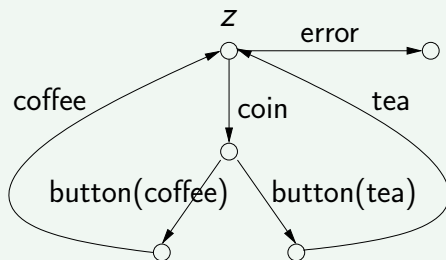


# Transition systems (examples)

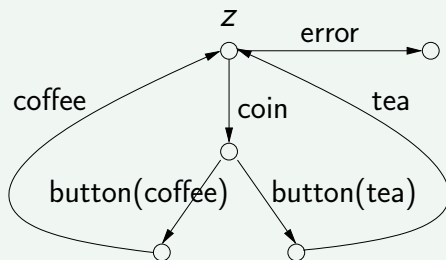


A machine that gives back change.

# Transition systems (examples)

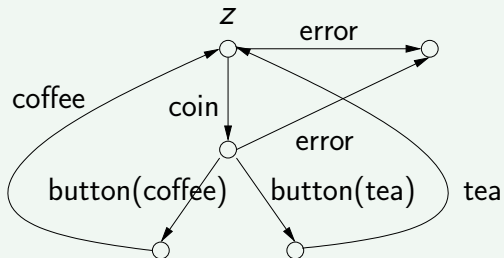


# Transition systems (examples)

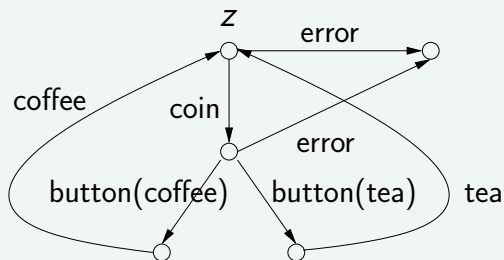


A machine with an error. The occurrence of an error is actually rather an internal action and could alternatively be modelled with a  $\tau$ .

# Transition systems (examples)

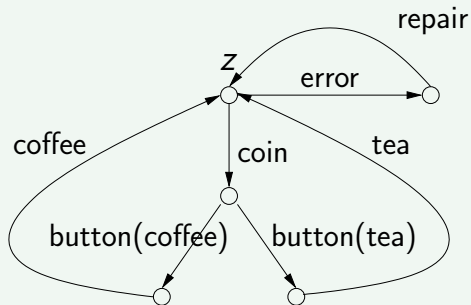


# Transition systems (examples)



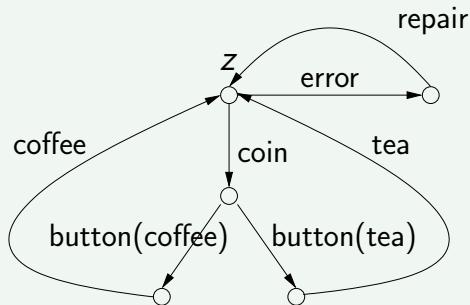
An (unfair) machine with faulty behaviour which may enter the error state after a coin has been inserted.

# Transition systems (examples)



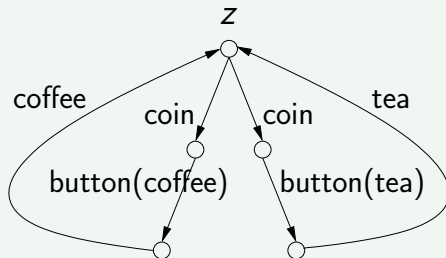


# Transition systems (examples)

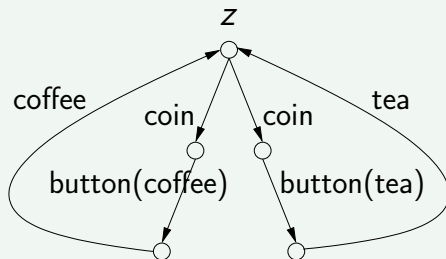


A machine with an error state that can be repaired.

# Transition systems (examples)



# Transition systems (examples)



A machine with non-deterministic behaviour that makes a choice of beverages for the user.

# Deterministic transition systems

## Deterministic transition system (definition)

A state  $s$  of a transition system is *deterministic*

$$\forall s', s'' \in S, a \in L (s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'') \implies s' = s''$$

A transition system induced by state  $s$  is deterministic if every reachable states from  $s$  is deterministic.

# Deterministic transition systems

## Deterministic transition system (definition)

A state  $s$  of a transition system is *deterministic*

$$\forall s', s'' \in S, a \in L (s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'') \implies s' = s''$$

A transition system induced by state  $s$  is deterministic if every reachable states from  $s$  is deterministic.

### Remarks:

- All tea/coffee-machines, apart from the last, are deterministic.

# Deterministic transition systems

## Deterministic transition system (definition)

A state  $s$  of a transition system is *deterministic*

$$\forall s', s'' \in S, a \in L (s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'') \implies s' = s''$$

A transition system induced by state  $s$  is deterministic if every reachable states from  $s$  is deterministic.

### Remarks:

- All tea/coffee-machines, apart from the last, are deterministic.
- Opposed to deterministic finite automata we do not require for deterministic transition systems that every action is feasible in every state.

## Some more definitions

- A state  $s$  of a transition system is a *deadlock* state iff

$$\forall a \in L \nexists t \ s \xrightarrow{a} t.$$

A transition system starting from  $s$  has a deadlock iff a deadlock state is reachable from  $s$ .

- A transition system is *regular* iff both its set of states and transitions are finite.
- A transition system is *image finite* iff each of its states has only finitely many outgoing transitions.