# 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 Ingólfsdóttir, 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.

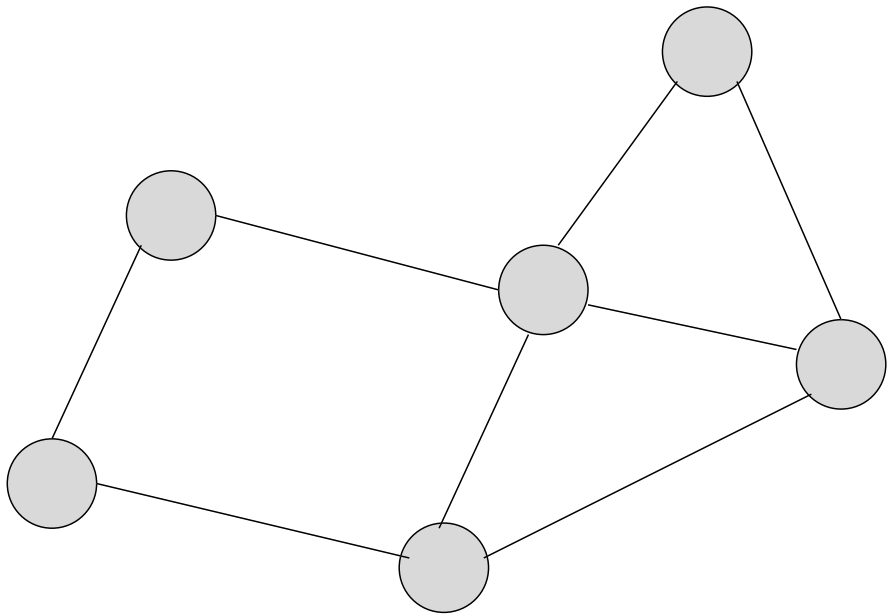
# Lecture style

- 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. print $x;$

1. $x = 1;$
2. $x = x \times 2;$
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. print $x;$

$\|$

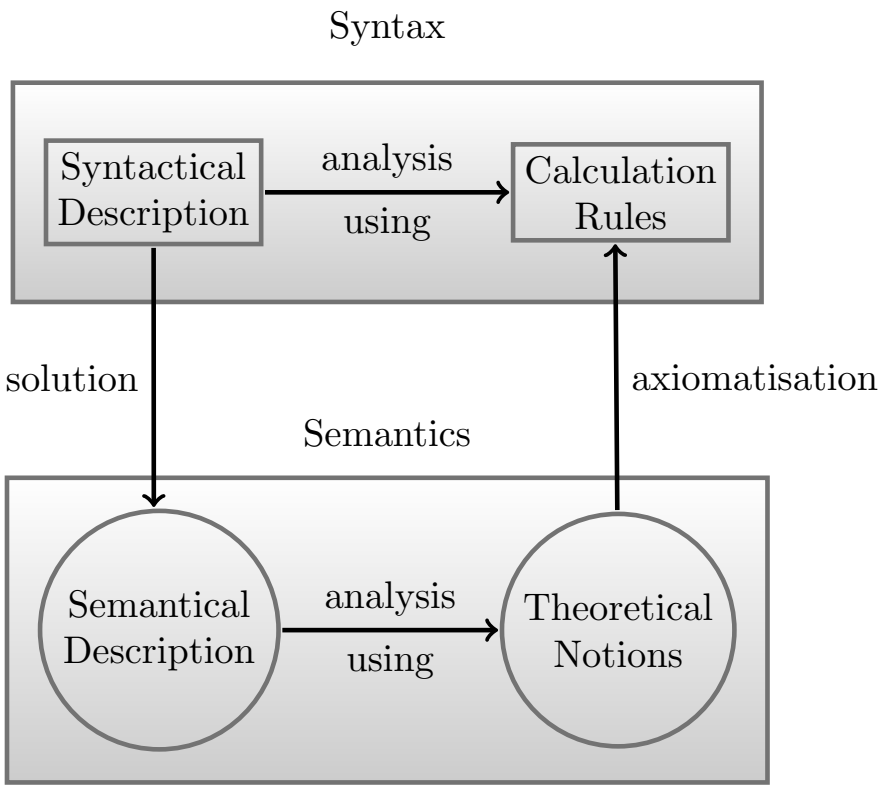
1. $x = 1;$
2. $x = x \times 2;$
3. print $x;$

# 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.

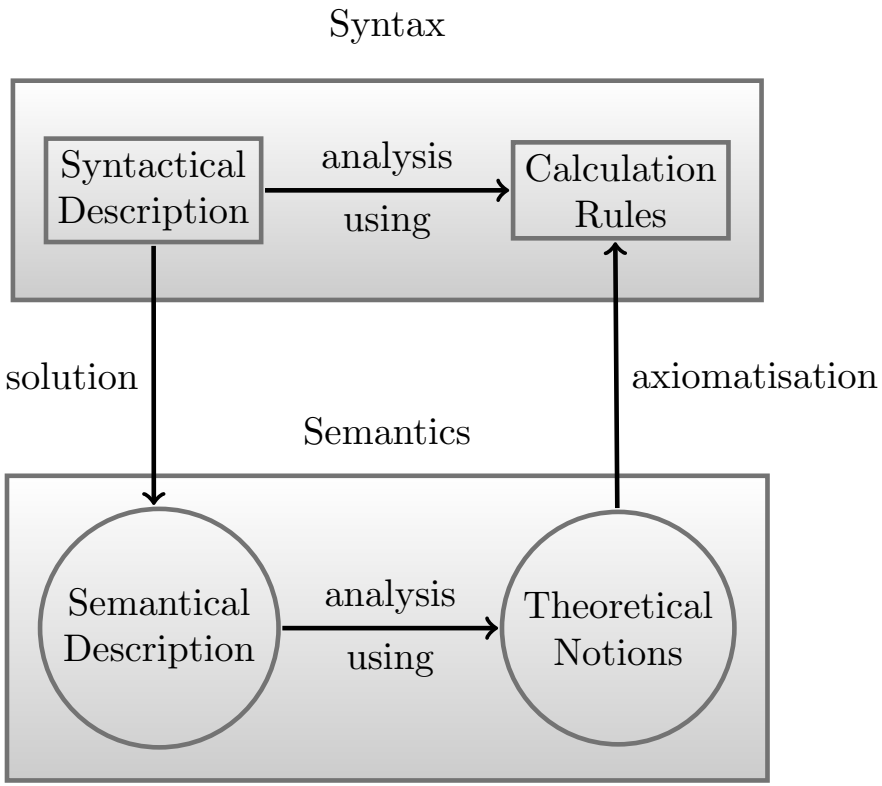
# Mathematical modelling (Cuijpers 2004.)

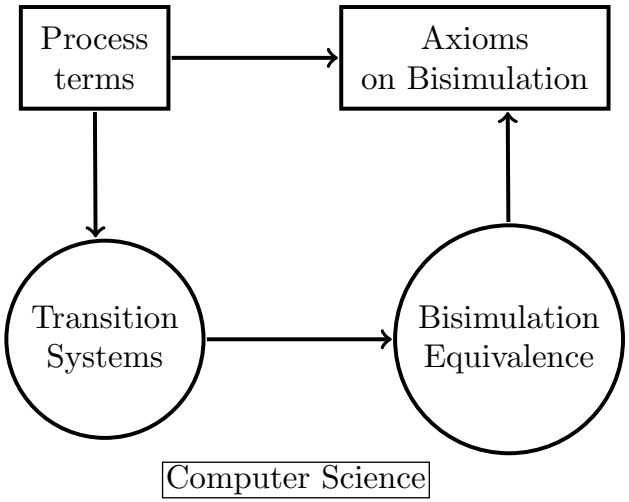
Syntax aims at concise and finite way of handling semantical notions, which can be infinite mathematical objects.



# Mathematical modelling (Cuijpers 2004.)

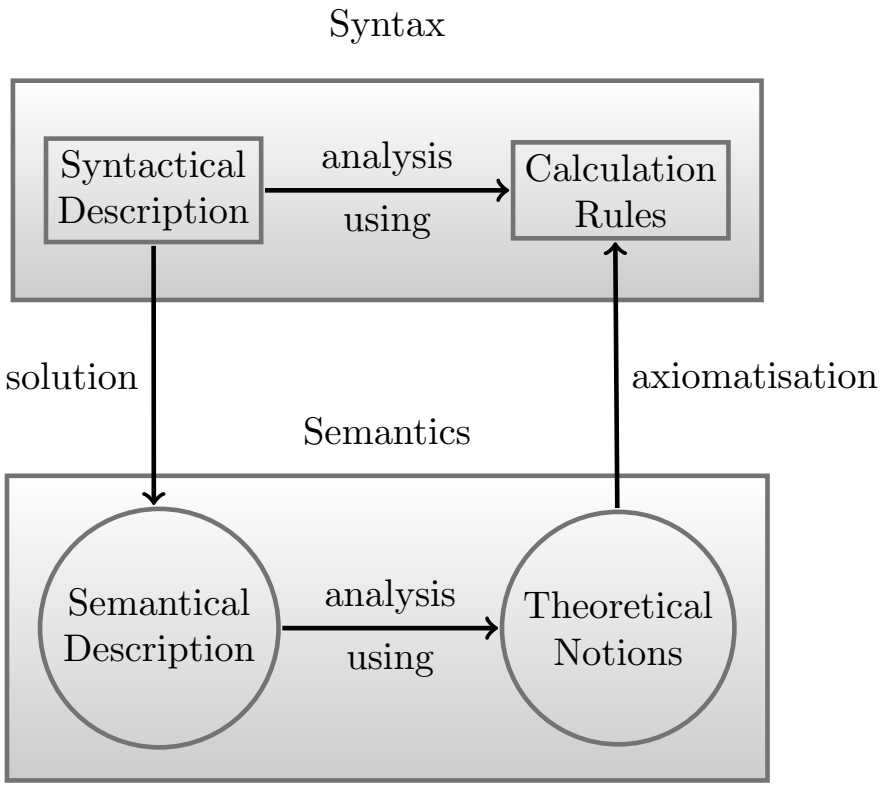
Syntax aims at concise and finite way of handling semantical notions, which can be infinite mathematical objects.

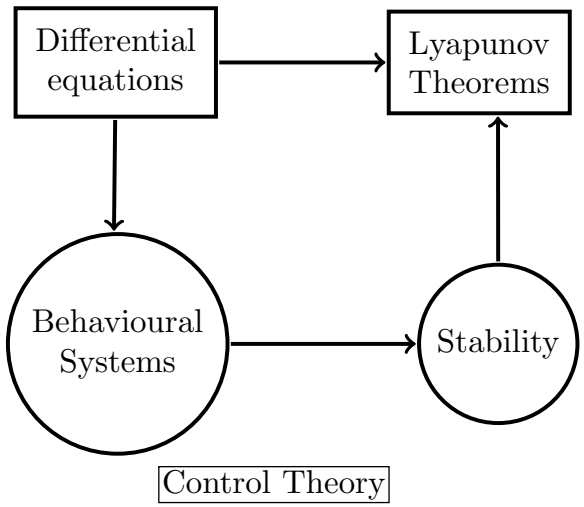


# Mathematical modelling (Cuijpers 2004.)

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.