Concurrency Theory

Lecture 1: Introduction

Stephan Mennicke
Knowledge-Based Systems Group

April 4, 2023

Organization

Lecture

Tuesday, DS 4 (13:00–14:30), APB E005
Exception: April 5 (tomorrow)

Exercise

Wednesday, DS 3 (11:10–12:40), APB E005

Website

`https://iccl.inf.tu-dresden.de/web/Concurrency_Theory_(SS2023)/en`

Exercise Sessions

- Proving technical results (sometimes even during lecture time)
- Formalizing definitions in LEAN
- Proving theorems and propositions in LEAN

LEAN THEOREM PROVER

Microsoft Research

Theorem Proving with LEAN

Course with SWS 0/0/4 (lecture/exercise/practical) in SS 2023

Lecturer

Lukas Gerlach
Stephan Mennicke

SWS

0/0/4

Modules

INF-B-510
INF-B-520
INF-MA-PR

Examination method

Seminar presentation

Matrix channel

#lean:tu-dresden.de

Information | Literature | Dates and Materials

Subscribe to events of this course (icalendar)

Practical	Kick-off Meeting	DS4, April 19, 2023 in APB 3027

Calendar

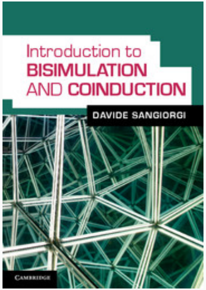
Monat | Woche | Tag

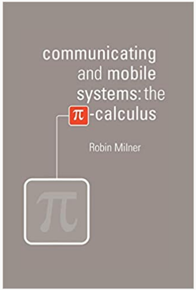
April 2023

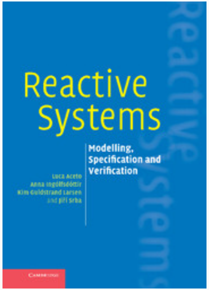
Heute

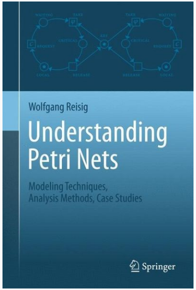
Mo	Di	Mi	Do	Fr	Sa	So
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Course Material









+ Lecture Notes regularly uploaded to the Website

Concurrency Theory

Course with SWS 2/2/0 (lecture/exercise/practical) in SS 2023

Lecturer

Stephan Mennicke

SWS

2/2/0

Modules

CMS-LM-ADV
CMS-LM-MOC
INF-B-510
INF-B-520
INF-BAS6
INF-PM-FOR
INF-VERT6
MCL-TCSL

Examination method

Oral exam

Information | Literature | Dates and Materials

Subscribe to events of this course (icalendar)

Lecture	Introduction	DS4, April 4, 2023 in APB 3027	File
Lecture	Towards Bisimulation	DS3, April 5, 2023 in APB E005	
Lecture	Coinduction: Examples, Duality Fixpoints	DS4, April 11, 2023 in APB E005	
No session	no exercise session	DS3, April 12, 2023 in APB E005	
Lecture	Coinduction: Proof Techniques	DS4, April 18, 2023 in APB E005	
Exercise	Introduction to LEAN	DS3, April 19, 2023 in APB E005	
Lecture	Algebraic Properties of Bisimulation	DS4, April 25, 2023 in APB E005	
Exercise	Formalizing Bisimilarity in LEAN	DS3, April 26, 2023 in APB E005	

Outline

- Introduce basic notions of concurrency theory
- Learn the features of common equivalence relations for concurrent processes
- Connect to other fields: computability and complexity theory
- Learn and apply the bisimulation proof method (coinduction)
- Study different synchronization paradigms to understand how concurrent (programming) languages are designed and analyzed

A Classic: Vending Machine

Consider a coffee/tea vending machine having a red color:

- you may insert money (say 1€)
- you may press the button for tea (req-tea) or the button for coffee (req-coffee)
- after pressing req-tea a tea beverage can be collected
- after pressing req-coffee a coffee beverage can be collected
- finally, the machine restarts its service

A Scenario

Your friend buys a similar vending machine, it is also red and behaves as follows:

- after inserting money (1€),
- the machine nondeterministically decides if the tea or coffee button can be pressed;
- and the respective beverage can be collected.

Of course, your friend is disappointed and would like to have a machine like you.

What is the difference? How can we describe the difference? How can we describe the specifications anyway?

Outline for First Lectures

1. How to formally describe the behavior of machines/systems?
2. What does "the same" behavior mean? What does it mean to have "different" behavior?
3. How to prove that two systems do not have the same behavior?

Answers:

1. Automata to the rescue: labeled transitions systems
2. Several answers, but the most important one: bisimulation
3. In case of bisimulation, coinduction

On Parallel Programs

Concurrent languages deal with language constructs to express that several program parts run in parallel (e. g., by an explicit parallel operator).

What is a parallel program?

Answer for *sequential programs*: functions.

Does the characterization lift to parallel programs?

```
X := 0
```

vs.

```
X := 1; X := X-1
```

Programs as Functions

P:

```
X := 0
```

Q:

```
X := 1; X := X-1
```

Viewed as functions, these two program snippets are the same function $f : (\mathbb{V} \to \mathbb{Z}) \to (\mathbb{V} \to \mathbb{Z})$ where $\mathbb{V}$ is the set of all variables like X: For a variable valuation $s : \mathbb{V} \to \mathbb{Z}$, $f(s) := s[\mathtt{X} \mapsto 0]$. Here, $s[\mathtt{X} \mapsto 0]$ is function s' with

$$s'(x) = \begin{cases} 0 & \text{if } x = \mathtt{X} \\ s(x) & \text{otherwise.} \end{cases}$$

In (denotational) semantics of programming languages we write

$$[\![P]\!] = [\![Q]\!] = f.$$

Issues with Functions

Lack of Compositionality

Suppose we use the following program context:

```
[.] | X := 0
```

Filling in P or Q for `[.]` makes a difference.

We say that the semantics is not compositional w. r. t. parallel composition.

Alternatively, program equality is not a congruence.

Termination Issues

Inherent Nondeterminism

What are Parallel Programs?

Parallel (or concurrent) programs are not functions, they are processes.

The question what a process actually is at the heart of concurrency theory.

Concurrency theory is the study of interacting processes and their (combined) behavior.

Key questions: When are two processes equal? When do they show the same behavior?

The two programs from before are distinguished by analyzing their interaction with the memory.

Therefore, a process formalism must allow for specifying when and how a process may interact with the outside world – also known as the environment.

Labeled Transition Systems (LTSs) – Definition and Notation

The most common formalism to study concurrent languages and, most importantly, their semantics is Labeled Transition Systems (LTSs).LTSs consist of

- states (or processes) and
- transitions between states.

Transitions are labeled by actions.

Definition 1 (LTS)
A labeled transition system is a triple $(Pr, Act, \rightarrow)$ where Pr is a set of *states* (or processes), Act is a set of *actions*, and $\rightarrow \subseteq Pr \times Act \times Pr$.

Instead of $(p, a, q) \in \rightarrow$ we often write $p \xrightarrow{a} q$. Likewise, $p \xrightarrow{a}$ means there is a $q \in Pr$ with $p \xrightarrow{a} q$ and $p \not\xrightarrow{a}$ means there is no such $q \in Pr$.

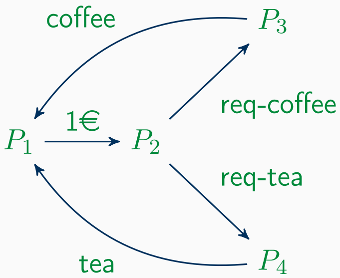
LTS – An Example

Consider the following LTS

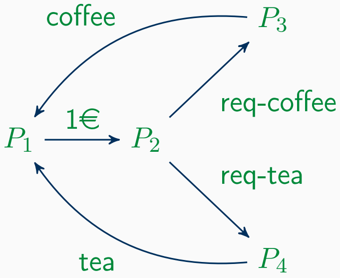
$$\mathcal{T} = (\{P_1, P_2, P_3, P_4\}, \{1€, \text{req-coffee}, \text{req-tea}, \text{coffee}, \text{tea}\}, \rightarrow)$$

with $P_1 \xrightarrow{1e} P_2, P_2 \xrightarrow{\text{req-coffee}} P_3, P_2 \xrightarrow{\text{req-tea}} P_4, P_3 \xrightarrow{\text{coffee}} P_1, P_4 \xrightarrow{\text{tea}} P_1$.

An LTS is usually depicted as a directed edge-labeled graph, called the *process graph*:



Wording



- Process P_1 has action 1€ enabled;
- P_1 performs action 1€ and, afterwards, behaves like P_2;

LTS – Further Notation

Definition 2
Given an LTS $\mathcal{T} = (Pr, Act, \rightarrow)$ and a process $P \in Pr$. The set of reachable states from P, $\mathit{Reach}(\mathcal{T}, P)$, is defined recursively:

- $P \in \mathit{Reach}(\mathcal{T}, P)$ and
- if $Q \in \mathrm{Reach}(\mathcal{T}, P)$ and $Q \xrightarrow{a} Q'$, then $Q' \in \mathit{Reach}(\mathcal{T}, P)$.

The LTS generated by P is the LTS $\mathcal{T}(P) = (\mathit{Reach}(\mathcal{T}, P), Act, \rightarrow')$ such that $\rightarrow' := \rightarrow \cap (\mathit{Reach}(\mathcal{T}, P) \times Act \times \mathit{Reach}(\mathcal{T}, P))$.

This allows us to speak about the *behavior of process* P (P is part of a bigger LTS).

LTS Classes

Definition 3
An LTS $(Pr, Act, \to)$ is

- image-finite if for each $a \in Act$ and each $p \in Pr$, the set $\{p' \in Pr \mid p \xrightarrow{a} p'\}$ is finite;
- finitely branching if for each $p \in Pr$, the set $\{p' \in Pr \mid \exists a \in Act : p \xrightarrow{a} p'\}$ is finite;
- finite-state if Pr is finite;
- finite if it is finite-state and acyclic;
- deterministic if for each $p \in Pr$, $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ imply $q = q'$.

These notions canonically carry over to processes.

Summary and Outlook

- Functions vs. processes
- LTSs for specification of process behaviors
- Misconception? Sequential formalism for process behaviors?

Next: P_1 vs. Q_1